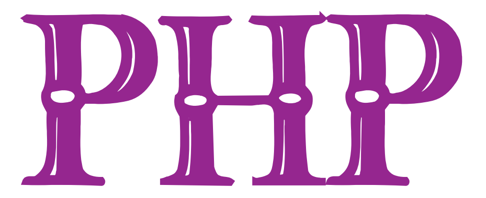
HACK X CRACK: TCP - LA ESENCIA DE LAS COMUNICACIONES POR RED



PROGRAMANDO SESIONES EN PHP

LOS IDENTIFICADORES DE SESION Y LAS COOKIES



PROGRAMANDO COOKIES !!!

3 SERVIDORES ON LINE PARA TUS PRACTICAS DE HACK

NÚMERO 20

LOS CUADERNOS DE HACK X CRACK

www.hackxcrack.com

HACK X CRACK - HACK X CRACK - HACK X CRACK

I.D.S.



SISTEMA DE DETECCION DE INTRUSOS !!!

* DETECTANDO PUERTOS EXTRAÑOS Y CONEXIONES SOSPECHOSAS
* REGLAS DE CABECERA DE CONTENIDO Y DE CONTROL
* CONTRAMEDIDAS EN TIEMPO REAL
* STREAM 4
* RPC DECODE
* PORTSCAN Y PORTSCAN 2

CURSO DE TCP/IP EL CORAZON DE INTERNET

TCP: PROTOCOLO DE CONTROL DE TRANSMISION

TCP FRENTE A UDP

CONTROL DE FLUJO

FLAGS

CONEXIONES VIRTUALES

CABECERAS TCP

¿POR QUÉ LAS APLICACIONES PREFIEREN TCP?



MENU X

MENUS A NUESTRO GUSTO Y COMO METER VARIOS JUEGOS EN 1 DVD

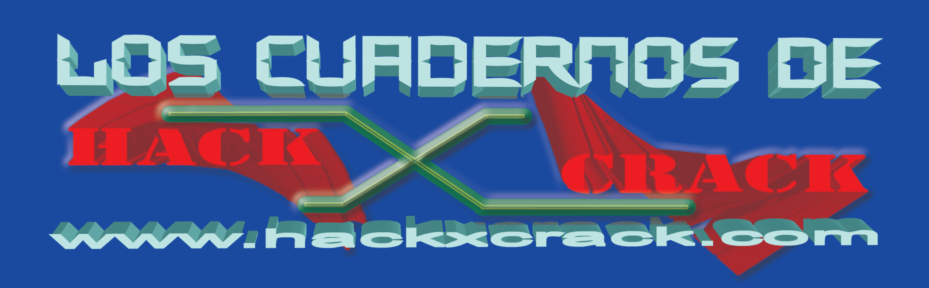
Nº 20 -- P.V.P. 4,5 EUROS



LOS MEJORES ARTÍCULOS GRATIS EN NUESTRA WEB

PC PASO A PASO: IDS - LA SEGURIDAD CON MAYUSCULAS !!!





EDITORIAL: EDITOTRANS S.L.
C.I.F: B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicación
PC PASO A PASO
Web: www.hackxcrack.com
Dirección: PERE MARTELL Nº 20, 2º - 1ª.
43001 TARRAGONA (ESPAÑA)

INDICE

INDICE DE ANUNCIANTES

Director de la Publicación
J. Sentís

E-mail contacto
director@hackxcrack.com

Diseño gráfico:
J. M. Velasco

E-mail contacto:
grafico@hackxcrack.com

Redactores
AZIMUT, ROTEADO, FASTIC, MORDEA, FAUSTO, ENTROPIC, MEIDOR, HASHIMUIRA, BACKBONE, ZORTEMIUS, AK22, DORKAN, KMORK, MAILA, TITINA, SIMPSIM... ... ...

Contacto redactores
redactores@hackxcrack.com

Colaboradores
Mas de 130 personas: de España, de Brasil, de Argentina, de Francia, de Alemania, de Japón y algún Estadounidense.

E-mail contacto
colaboradores@hackxcrack.com

Imprime
I.G. PRINTONE S.A. Tel 91 808 50 15

DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

TELÉFONO DE ATENCIÓN AL CLIENTE: 977 22 45 80
Petición de Números atrasados y Suscripciones (Srta. Genoveva)
HORARIO DE ATENCIÓN: DE 9:30 A 13:30
(LUNES A VIERNES)

NUMERO 20 -- PRINTED IN SPAIN
PERIOCIDAD MENSUAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756

CURSO DE PHP
APRENDE A MANEJAR SESIONES

Continuamos con el curso de PHP y en esta entrega aprenderemos a manejar sesiones. He decidido explicar las sesiones antes de seguir con socket, ya que con el manejo de las sesiones se comprende como un portal puede dar acceso a partes privadas.

Existen varias formas de mantener la sesión del navegante, se explicarán algunas, las más conocidas y que encontrareis en la mayoría de portales.

1. Introducción de session

La sesión va unida al usuario, se pueden crear variables a nivel de sesión que son accesibles desde cualquier página siempre que la sesión este activa.

Una sesión se crea cuando un usuario accede a una página web, a partir de ese momento todos los accesos a las páginas de ese sito web pueden seguirse a través de la sesión creada. Cada sesión creada lleva asociado un identificador, que identificará esa sesión con el usuario que provocó su creación.

Además, las variables definidas a nivel de sesión permanecen accesibles desde todas las páginas php del site, durante toda la sesión del usuario, por lo tanto son una especie de variables globales que permanecen activas mientras la sesión esté activa.

Una sesión estará activa siempre que el usuario se comunique con el servidor web, cambie de una página web a otra, envíe un formulario, etc. Cada sesión tiene un tiempo máximo de inactividad permitida (lo normal suele ser 20 minutos, pero se puede cambiar ese valor), si un usuario permanece sin actividad durante un periodo de tiempo determinado o abandona el sitio Web, todas sus variables e identificadores de sesión son borrados.

Dentro del fichero *php.ini*, modificando el valor de la variable **session.cookie_lifetime** se modifica el tiempo de vida (o tiempo de inactividad) de la sesión.

La variable **session.cookie_lifetime** especifica la duración de la cookie en segundos, que se manda al navegador. El valor 0 significa "hasta que se cierra el navegador", y es el que se encuentra por defecto.

Un ejemplo muy claro de esto es la banca electrónica. Si nos identificamos en nuestro banco on-line para consultar el saldo de nuestra cuenta bancaria y nos vamos a tomar un café, seguramente, cuando volvamos el banco no nos dejará realizar más operaciones hasta que nos volvamos a identificar en el sistema.

Otro ejemplo de la utilización de las sesiones es el comercio electrónico, cuando compramos vamos almacenando productos en el carrito de la compra y, aunque cambiemos de página web, los productos siguen estando en el carrito. Esto es gracias a que la información relativa a esos productos está almacenada en variables de sesión.

Existen dos formas de mantener y continuar una sesión mientras el usuario navega por las páginas web. En **la primera** de ellas es mediante el uso de las *cookies* (datos guardados en el cliente) y en **la segunda**, el identificador de la sesión se incluye en la URL de la página como un parámetro más. El módulo de gestión de sesiones que proporciona PHP permite utilizar ambas formas.

El módulo de sesiones admite ambas formas. Las Cookies son la mejor opción, pero como no son fiables (los clientes no están obligados a aceptarlas), no podemos confiar en ellas. El segundo método incrusta el "session id" directamente en las URLs.

Las sesiones (y toda la información relativa a ellas) en PHP se guardan ficheros que están en un directorio. Este directorio está especificado en la variable **session.save_path** dentro del apartado [Session] del fichero *php.ini*.

Todas las variables y parámetros necesarios para el correcto funcionamiento de las variables se encuentran en el fichero *php.ini*.

Tanto para leer como para escribir variables de sesión utilizaremos el array asociativo $HTTP_SESSION_VARS[var] o $_SESSION[var] que es su versión abreviada.

Vamos a ver el uso de las sesiones "plan rápido" con un ejemplo ejemplo:

1)- En primer lugar tenemos el código de una página PHP (llamemosla php7.php) que guarda las variables **nombre**, **apellidos** y **sessionId** en la sesión. Ya sabes, como siempre copia el código en el Block de Notas y guardalo con el nombre php7.php

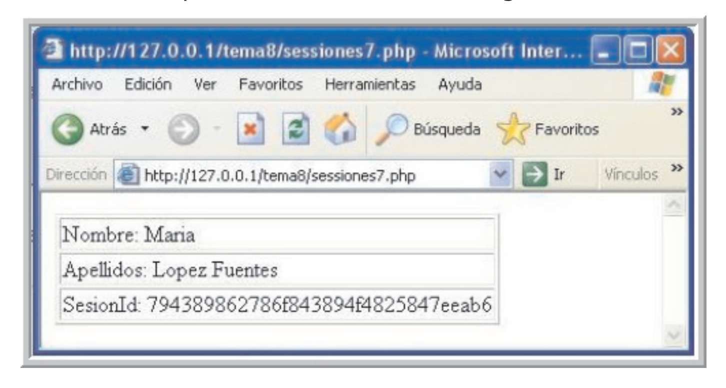
```
<?php session_start();?>
<HTML>
<HEAD>
<TITLE>Sesion Paso 1</TITLE>
</HEAD>
<BODY>
<?php
  $nombre = "Maria";
  $apellidos = "Lopez Fuentes";
  $sesionId = session_id();
  $_SESSION[nombre]=$nombre;
  $_SESSION[apellidos]=$apellidos;
  $_SESSION[sesionId]=$sesionId;
?>
<a href="sessiones7.php">Siguiente</a>
</BODY>
</HTML>
```

- En segundo lugar tenemos otra página (sessiones7.php) donde recogeremos el valor de dichas variables almacenadas en la sesión. Ya sabes, como siempre copia el código en el Block de Notas y guardalo con el nombre php7.php

```
<?php  session_start();  ?>
<HTML>
<HEAD>
<TITLE>Sesion Paso 2</TITLE>
</HEAD>
<BODY>
 <TABLE BORDER="1">
  <TR>
<TD>Nombre: <?= $_SESSION["nombre"]?></TD>
  </TR>
  <TR>
    <TD>Apellidos: <?= $_SESSION["apellidos"]?></TD>
  </TR>
  <TR>
    <TD>SesionId: <?= $_SESSION["sesionId"]?></TD>
  </TR>
 </TABLE>
</BODY>
</HTML>
```

Para que se almacenen las variables en la sesión primero debemos ejecutar el código de la primera página PHP (php7.php) y a continuación ejecutaremos la segunda página. Si no siguiéramos esta secuencia y ejecutásemos directamente el código de la segunda página, las variables de la sesión no tendrían ningún valor.

El resultado que obtendremos será el siguiente:



Visto el ejemplo vamos a ver el tema con más detalle.

2. Funciones

2.1 session_start

Esta función crea una nueva sesión o continúa con la sesión actual.
La sintaxis de la función es la siguiente:

```
session_start();
```

Todas las páginas PHP que utilicen sesiones deben incluir esta función que indica al intérprete PHP que recupere toda la información relativa a las sesiones, para posteriormente hacer uso de ella.

Esta función debe escribirse antes de que se escriba cualquier carácter en el cliente, incluso antes de las cabeceras HTML, si no es así se producirá un error. Para evitar problemas es mejor colocar esta función al principio de nuestra página PHP.

2.2 session_destroy

Esta función elimina todos los datos asociados a una sesión.
La sintaxis de la sesión es la siguiente:

session_destroy();

Está función no borra la sesión ni la cookie donde se encuentra almacenada, sólo borra los datos asociados a la misma, es decir, borra las variables que se encuentran asociadas a una sesión.

2.3 session_name

Esta función devuelve el nombre de la sesión utilizada en ese momento. Por defecto, este valor es "PHPSESSID" y esta definido dentro del fichero php.ini. Si se desea especificar un nombre diferente a la sesión, éste se deberá pasar como parámetro de la función.

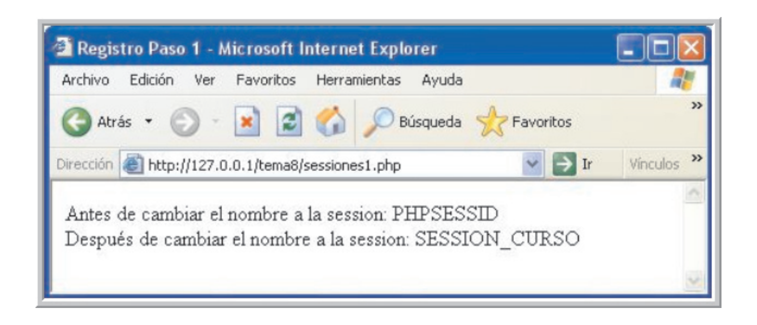
La sintaxis de la función es la siguiente:

session_name(nombreSesion);

El parámetro nombreSesion es opcional.

```
<?php
print("Antes de cambiar el nombre a la session: " . session_name() . "<br>");
session_name("SESSION_CURSO");
print("Después de cambiar el nombre a la session: " . session_name() . "<br>");
?>
```

Y el resultado obtenido al ejecutar el código anterior es:



2.4 session_module_name

Devuelve el valor de la variable **session.save_handler** situada en el fichero *php.ini* encargada de definir el tipo de controlador utilizado para grabar y recuperar el manejo de las sesiones que por defecto es *files.*

La sintaxis de la función es la siguiente:

session_module_name(handler);

El parámetro handler es opcional.

2.5 session_save_path

Permite conocer la ruta en la que se guardan las sesiones si el valor de session.save_handler es *files.* En la mayoría de los casos se trata de un directorio temporal.

Esta función recupera y modifica el valor de la variable **session.save_path** del fichero php.ini.

Si session_save_path apunta a un directorio con permiso de lectura por el resto de usuarios, como */tmp* (la opción por defecto), los demás usuarios del servidor pueden conseguir robar las sesiones obteniéndolas de la lista de archivos de ese directorio.

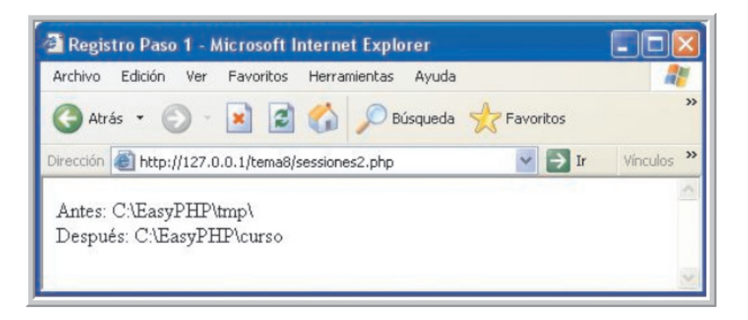
La sintaxis de la función es la siguiente:

session_save_path(path);

El parámetro path es opcional.

```
<?php
print(session_save_path());
session_save_path("C:\EasyPHP\curso");
print(session_save_path());
?>
```

Y el resultado obtenido es:

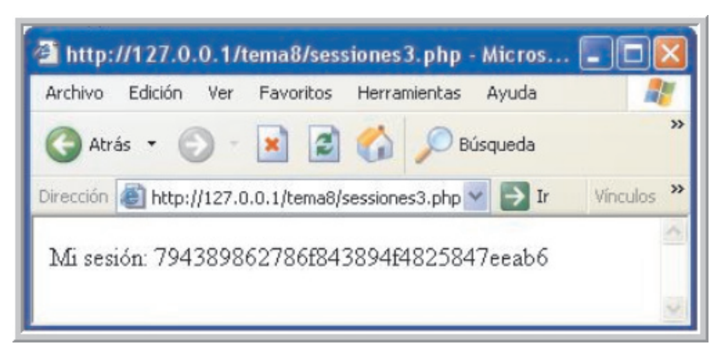


2.6 session_id

Esta función devuelve el identificador de la sesión del usuario en el sitio Web, por lo tanto es una de las funciones más importantes en el manejo de la sesión.

```
<?php
session_start();
print("Mi sesión: " . session_id() .
"<BR>");
?>
```

Y el resultado obtenido es:



2.7 session_register

Esta función se encarga de registrar los parámetros que se le pasan a la función, como variables a nivel de sesión. Por lo tanto el número de argumentos de esta función no es fijo y dependerá de las variables que deseemos declarar.

Para explicar el funcionamiento de la función *session_register()* lo veremos en el siguiente ejemplo (**sesion1**):

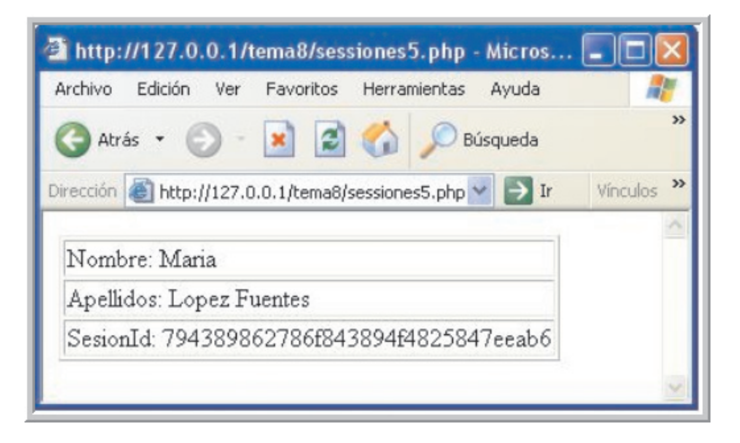
```
<?php session_start();?>
<HTML>
<HEAD>
<TITLE>Sesion Paso 1</TITLE>
</HEAD>
<BODY>
<?php
  $nombre = "Maria";
  $apellidos = "Lopez Fuentes";
  $sesionId = session_id();
  session_register("nombre", "apellidos", "sesionId");
?>

<a href="sessiones5.php">Siguiente</a>
</BODY>
</HTML>
```

Y recogemos los valores en la siguiente página (sesion2), que guardaremos como sessioones5.php:

```
<?php  session_start();  ?>
<HTML>
<HEAD>
<TITLE>Sesion Paso 2</TITLE>
</HEAD>
<BODY>
  <TABLE BORDER="1">
    <TR>
      <TD>Nombre: <?= $GLOBALS["nombre"]?></TD>
    </TR>
    <TR>
      <TD>Apellidos: <?= $_SESSION["apellidos"]?></TD>
    </TR>
    <TR>
      <TD>SesionId: <?= $HTTP_SESSION_VARS["sesionId"]?></TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

Y el resultado obtenido es:



Los valores de las variables almacenadas en la sesión se recogen utilizando $HTTP_SESSION_VARS[nombre_variable] pero también se puede utilizar su versión reducida $_SESSION, tal como podemos ver en el ejemplo anterior.

Como podemos observar la recogida de los valores de las variables también la podemos realizar mediante el array $GLOBALS que guarda todas las variables de ámbito global.

Si no se llamase la función session_start() para continuar con la sesión, no se podrían recuperar las variables globales.

Si nuestro script usa **session_register()**, no funcionará en entornos en donde la directiva PHP *session.register_globals* de php.ini esté deshabilitada. Actualmente el uso de session.register() está desfasado, por ello tanto para guardar como para recoger las variables de la sesión utilizaremos $_SESSION como ya se ha visto anteriormente.

2.8 session_unregister

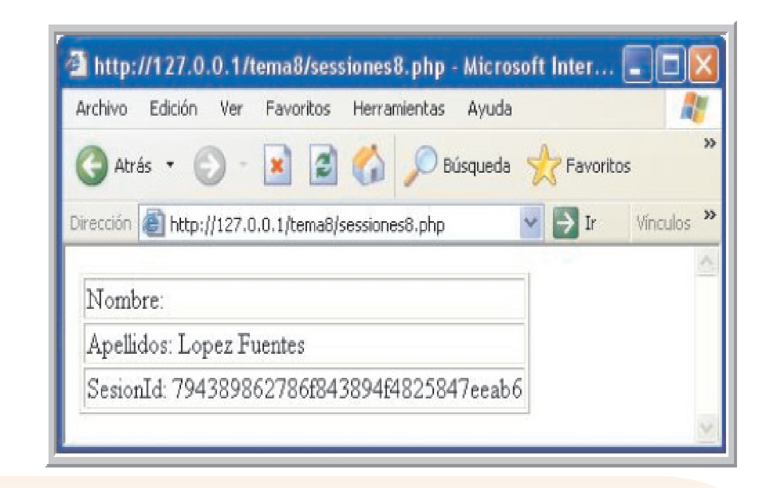
Esta función realiza el proceso contrario de la función anterior (session_register). Se encarga de eliminar la relación entre la variable y la sesión. Los argumentos de esta función son los nombres de las variables que queremos liberar.

Si sustituimos el código de *sesion2* por el siguiente:

```
 <?php
    session_start();
    session_unregister($nombre);
 ?>
<HTML>
<HEAD>
<TITLE>Sesion Paso 2</TITLE>
</HEAD>
<BODY>
 <TABLE BORDER="1">
   <TR>
     <TD>Nombre: <?= $GLOBALS["nombre"]?></TD>
   </TR>
   <TR>
     <TD>Apellidos: <?= $_SESSION["apellidos"]?></TD>
   </TR>
   <TR>
     <TD>SesionId: <?= $HTTP_SESSION_VARS["sesionId"]?></TD>
   </TR>
</TABLE>
</BODY>
</HTML>
```

Recuerda que debemos ejecutar antes el código de **sesion1** y después el de **sesion2**

Obtendríamos el siguiente resultado:



Para vaciar el valor de una variable de sesión también lo podemos realizar como en cualquier otra variable utilizando la función *unset(nombre_variable)* (Definida en el tema1)

2.9 session_is_registered

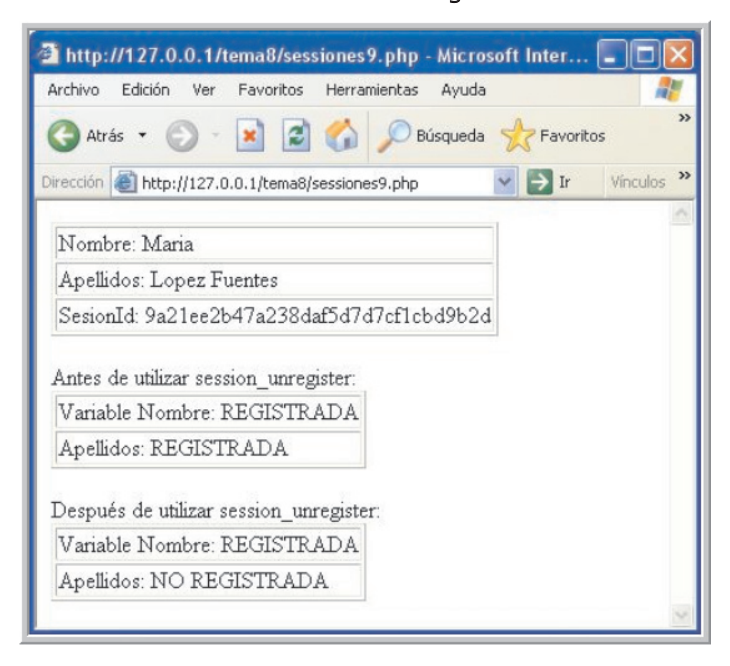
Esta función nos permite averiguar si una variable ya ha sido registrada en la sesión. La función devolverá *true* si la variable está registrada en la sesión y *false* en caso contrario.

Al sustituir el código de **sesion2** por el siguiente código:

```
<?php
 session_start();
?>
<HTML>
<HEAD>
<TITLE>Sesion Paso 2</TITLE>
</HEAD>
<BODY>
  <TABLE BORDER="1">
    <TR>
      <TD>Nombre: <?= $GLOBALS["nombre"]?></TD>
    </TR>
    <TR> <TD>Apellidos: <?= $GLOBALS["apellidos"]?></TD>
    </TR>
    <TR> <TD>SesionId: <?= $GLOBALS["sesionId"]?></TD>
    </TR>
</TABLE>
<br>
Antes de utilizar session_unregister:<br>
<TABLE BORDER="1">
    <TR> <TD>Variable Nombre:
    <?php
      if  (session_is_registered("nombre")) {?>
            REGISTRADA
    <?php
      } else {?>
            NO REGISTRADA
<?php
}?>       </TD>
</TR>
<TR>       <TD>Apellidos:
<?php
if  (session_is_registered("apellidos")) {?>
          REGISTRADA
<?php
 } else {?>
          NO REGISTRADA
<?php
}?>       </TD>
</TR>
</TABLE>
<?php
 session_unregister("apellidos");?>
<br>
Después de utilizar session_unregister:<br>
<TABLE BORDER="1">
  <TR>       <TD>Variable Nombre:
  <?php if  (session_is_registered("nombre")) {?>
            REGISTRADA
  <?php } else {?>
            NO REGISTRADA
  <?php
  }?>       </TD>
  </TR>
  <TR>       <TD>Apellidos:
   <?php if  (session_is_registered("apellidos")) {?>
            REGISTRADA
  <?php } else {?>
            NO REGISTRADA
  <?php }?>
  </TD>
 </TR>
</TABLE>
</BODY>
</HTML>
```

Recuerda que debemos ejecutar antes el código de **sesion1** y después el de **sesion2**

Obtendremos el siguiente resultado:



Para comprobar si variable de sesión ha sido inicializada, también lo podemos realizar como en cualquier otra variable utilizando la función *isset(nombre_variable)* (Definida en el tema1)

2.10 session_decode

Recupera todos los datos de la sesión que están almacenados en una cadena y que es pasada como argumento, dando valores a las variables utilizadas en la sesión.

La sintaxis de la función es la siguiente:

session_encode(string datos);

2.11 session_encode

Codifica los datos que provienen de la sesión actual, en una variable de tipo string.

La sintaxis de la función es la siguiente:

session_encode();

```
<?php session_start();
print(session_encode());
?>
```

3. Cookies

Uno de los problemas que nos encontramos en Internet es no poder identificar a los usuarios que visitan nuestras páginas. Si un usuario abandona nuestro site y a los cinco minutos vuelve, el servidor no es capaz de saber si se trata del mismo usuario o de otro diferente. No nos vale controlar esto con las sesiones, ya que si el usuario cerrase su navegador o reiniciase su equipo, estaría en una sesión distinta y le habríamos perdido la pista. Otra manera de mantener la sesión identificando al navegante es utilizando las cookies.

Para solucionar este problema utilizaremos las cookies, que son pequeños ficheros de texto que el servidor deposita en el cliente (es un espacio que reserva el navegador) para que, si posteriormente el mismo usuario accede al servidor, éste es reconocido por el servidor al examinar la cookie previamente depositada.

El pequeño fichero de texto que forma una cookie se guarda de la siguiente forma:
nombre=valor
y, a pesar de que se trata de algo inofensivo, están rodeadas de una seguridad adicional:
un navegador web como Internet Explorer o Netscape, **tan sólo puede alamacenar un número máximo de 300 cookies en total y no más de 20 cookies por servidor**.

Además el tamaño de una cookie está limitado a 4kb. Si se alcanzase el límite de 300 cookies, estas se irían eliminando por antigüedad.

El uso de las cookies es muy habitual en las Webs en los que debe existir un seguimiento de las acciones del usuario.

Las cookies son parte de la cabecera HTTP, por tanto debemos llamar la función **setcookie()** (utilizada para crear cookies) antes de que se produzca cualquier salida al navegador. Cualquier cookie del cliente, automáticamente se convertirá en una variable PHP igual como ocurre con los métodos de datos GET y POST, dependiendo de las variables de configuración *register_globals* y *variables_order*.

En PHP 4.1.0 y posteriores, la matriz auto-global $_COOKIE será siempre actualizada con cualquier cookie mandada por el cliente.

$HTTP_COOKIE_VARS también se actualiza en versiones anteriores de PHP cuando la variable de configuración track_vars esté activada. (Siempre está activada a partir de PHP 4.0.3.)

3.1 Crear una Cookie

Para trabajar con cookies, lo primero que debemos hacer es crear una cookie y para ello utilizaremos la función *setcookie*

La sintaxis de la función es la siguiente:

```
int setcookie (string nombre, string
valor, int caducidad, string ruta,
string dominio, int secure)
```

Todos los parámetros excepto *nombre* son opcionales. También podemos sustituir cualquier parámetro por una cadena de texto vacía ("") y saltar así ese parámetro.

Los parámetros *caducidad* y *secure* son números enteros y no podemos sustituirlos con una cadena de texto vacía, en su lugar debemos utilizar un cero (0).

El parámetro *caducidad* es un entero de tiempo típico de UNIX tal como lo devuelven las funciones **time()** o **mktime()**.

El parámetro *secure* indica que la cookie se debe transmitir única y exclusivamente sobre una conexión segura HTTPS.

```
<?php
$usuario="valor";
//Crea una cookie
 setcookie("nombre", $usuario);
//Crea una cookie que caducará en una hora
 setcookie("nombre", $usuario, time() + 3600);
//Crea una cookie que caducará el 31/12/2005
 setcookie("nombre", $usuario, mktime(10,20,0,12,31,2004));
?>
```

Si deseamos asignar múltiples valores a una cookie simple, añade simplemente [] a el nombre de la cookie, es decir, convertir a la cookie en un array de cookies.

```
<?php
setcookie( "nombre[3]", "David" );
setcookie( "nombre[2]", "Sabina" );
setcookie( "nombre[1]", "Maria" );
if ( isset( $nombre ) ) {
   while( list( $name, $value ) = each( $nombre ) ) {
      print("$name == $value<br>");
   }
}
?>
```

Al ejecutar el código anterior obtenemos el siguiente resultado:



setcookie() define una cookie para ser enviada con el resto de la información de la cabecera HTTP.

Las cookies deben enviarse *antes* de mandar cualquier otra cabecera (esta es una restricción de las cookies, no de PHP). Esto requiere que situemos las llamadas a esta función antes de cualquier etiqueta <html> o <head>.

Si escribimos cualquier cosa antes de la creación de la cookie la ejecución de nuestra página PHP generará un error.

3.2 Borrar una Cookie

Para borrar una cookie se utiliza el mismo sistema que para crear una nueva, pero no se le asocia ningún valor.

```
<?php
//Borra una cookie
 setcookie("nombre");
?>
```

De esta forma se elimina la cookie del cliente.

3.3 Leer una Cookie

Para acceder al contenido de una cookie, basta con utilizar el nombre de la variable.

```
<?php
//Leemos el contenido de una cookie
 print("\$alumno:" . $ alumno);
?>
```

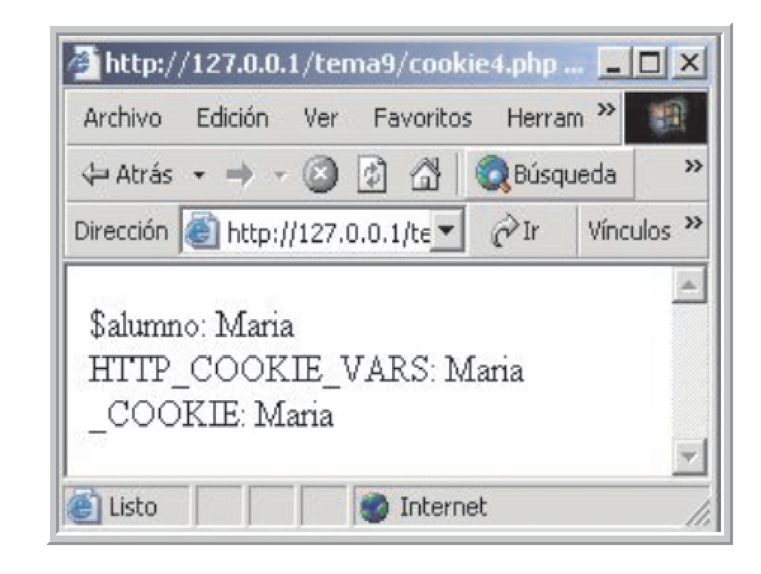
También podemos acceder al valor de la cookie utilizando: $HTTP_COOKIE_VARS["nombre"].

```
<?php
//Leemos el contenido de una cookie
 print("HTTP_COOKIE_VARS: " . $HTTP_COOKIE_VARS["alumno"]);
?>
```

O podemos utilizar su forma abreviada.

```
<?php
//Leemos el contenido de una cookie
 print("$_COOKIE: " . $_COOKIE["alumno"]);
?>
```

El resultado de ejecutar los tres ejemplos anteriores es:



3.4 Limitar el alcance de las cookies

Las cookies se pueden limitar por medio de los parámetros que utilizamos en su creación, la limitación de una cookie se realiza por tres ámbitos:

- **directorio de acceso:** delimita el directorio de acceso a la cookie, solo en el directorio que le indiquemos podremos tener acceso a la cookie. Se define en el parámetro *ruta* en la creación de la cookie.
- **fecha de caducidad:** estable la fecha de caducidad de la cookie, se le debe indicar por medio de un número entero, utilizando las funciones mktime() o time(). Se define en el parámetro *caducidad* en la creación de la cookie.
- **dominio de procedencia:** sólo puede accede a esa cookie el dominio indicado en su creación. Se define en el parámetro *dominio* en la creación de la cookie.

4 Enviar cabeceras HTTP

Utilizaremos la función *header()* para enviar cabeceras http. Su sintaxis es la siguiente:

```
header("Nombre Cabecera: valor cabecera");
```

```
<?php
//Redirige a una página automáticamente
 header("Location: index.php");
//No cachea la página que contiene esta función
 header("Cache-Control: no-cache, must-revalidate");
//Tipo de documento que se envía al cliente
header("Content-Type: text/html");
?>
```

Al igual que la función setcookie se debe utilizar esta función al principio de la página, antes de escribir cualquier valor en el cliente.

En la siguiente entrega haremos uso de todo lo aprendido aplicado y seguiremos con los sockets.

CURSO DE TCP IP: (3ª ENTREGA) TCP (TRANSMISION CONTROL PROTOCOL) - 1ª PARTE.

- Empezaremos a estudiar las CABECERAS TCP
- Veremos las diferencias entre TCP y UDP
- ¿Qué es eso de "protocolo orientado a la conexión"? ---> conexiones virtuales

1. INTRODUCCION

Después de haber estado hablando indirectamente sobre el protocolo TCP/IP durante 10 meses en la serie RAW, al fin vamos a entrar de lleno en la base de todo esto: la base sobre la que se apoyó toda la serie RAW.

Al hablar del protocolo TCP al fin vamos a comprender qué es lo que ocurre realmente cuando hacemos un Telnet y la necesidad de establecer conexiones de este tipo.

Comprenderemos también por qué se utiliza TCP como protocolo de transporte para la mayoría de las aplicaciones, al compararlo con UDP y comprobar que son muchas las ventajas.

Con este artículo, entramos a fondo en conceptos nuevos, como son las conexiones virtuales, el control de flujo, los flags, etc. Quizá no os queden del todo claros los conceptos después de terminar de leerlo, pero no os preocupéis, que está todo previsto.

El curso continuará con una segunda entrega dedicada a TCP, en la cual aclararemos muchos de los conceptos que quedarán hoy en el aire, y veremos una serie de ejemplos que conseguirán con la práctica lo que no se puede conseguir con la mera teoría.

Además, para hacerlo un poco más amena, la segunda entrega incluirá también algunas técnicas de hacking basadas en el funcionamiento de TCP, que es lo que a muchos de vosotros más os interesa. ;-) Aprenderemos además a construir las cabeceras tal y como son en realidad, en binario. Pero lo más importante es que podremos ver en detalle cómo es el tráfico que circula constantemente entre nuestro ordenador y la red Internet.

2. DIFERENCIAS ENTRE TCP Y UDP

Por si no lo habéis hecho, es importante que antes de continuar repaséis los dos artículos anteriores del curso de TCP/IP (Introducción, y UDP), ya que vamos a empezar comparando este nuevo protocolo con el que ya conocíamos, el protocolo UDP.

2.1. Conexiones virtuales

Volvemos a insistir una vez más en que una de las principales diferencias entre TCP y UDP es la existencia de conexiones virtuales. Como ya vimos, el protocolo TCP se dice que es **orientado a conexión**

(en realidad conexiones virtuales), al contrario que UDP (no orientado a conexión).

Las implicaciones de esto son muchas, tal y como iremos viendo a lo largo del artículo.

De momento quedémonos con la idea de conexión virtual.

¿Por qué se habla de conexiones virtuales, y no de conexiones a secas? Para encontrar una respuesta a esta pregunta basta con pensar un poco en la arquitectura de la red Internet (y de cualquier red TCP/IP).

Si Internet hubiese existido hace 50 años, probablemente su arquitectura hubiera sido muy diferente, igual que la red telefónica de ahora es muy diferente a la que había entonces. La red telefónica de aquellos años dependía de una figura humana bien conocida: la **telefonista**.

Las telefonistas eran unas mujeres casi biónicas que actuaban como intermediarias entre los distintos clientes de la red telefónica para que estos pudieran comunicarse entre sí. Cuando un cliente deseaba hablar con otro, una telefonista tenía que establecer una conexión manual entre las líneas de ambos clientes, conectando los cables correspondientes en un panel de conexiones.

Imaginemos cómo habría sido la Internet en aquellos años. Un autentico ejército de "internetistas" trabajando 24 horas para dar servicio a todas las conexiones que estableciese cualquier cliente: cada vez que alguien desease leer su correo electrónico, una internetista tendría que establecer una conexión en el panel entre el cliente y su servidor de POP3, cada vez que alguien visitase una página web una internetista tendría que establecer una conexión entre el cliente y el servidor de la página web, etc., etc.

Desde luego, esto habría sido una locura inviable, sin contar con el hecho de que la red sería muy lenta y muy insegura.

Si tratásemos de mejorar esta idea sustituyendo a las internetistas humanas por un medio mecánico automatizado que estableciese las conexiones mediante relés seguiríamos teniendo muchos problemas, y uno de los más preocupantes sería el de la velocidad.

Se mire por donde se mire, una red en la que haya que establecer nuevas conexiones cada vez que se desee utilizar un servicio siempre será más lenta que una red en la que todo esté conectado de antemano.

Y precisamente esa es la topología de la red Internet: todo está conectado de antemano. Nuestro PC siempre tiene un único cable que nos conecta con Internet (el cable de red que va al router, o el cable que va al modem, o el "cable" etéreo que nos une con nuestra estación wireless). Y lo mismo ocurre con cualquier máquina conectada a Internet. Nunca hay que cambiar ningún cable de sitio, por muy diferentes que sean las conexiones que queramos establecer. Tanto si vamos a enviar un correo a

nuestro mejor amigo, como si vamos a ver una página web de Japón, las conexiones físicas son siempre las mismas.

Esto hace que los protocolos de la red Internet tengan necesariamente que estar muy bien pensados para que esto no de lugar a un completo caos. Para evitar estas confusiones nace el concepto de **conexión virtual**, que busca las ventajas de las redes en las que las conexiones no están establecidas de antemano, pero sin toparse con sus desventajas.

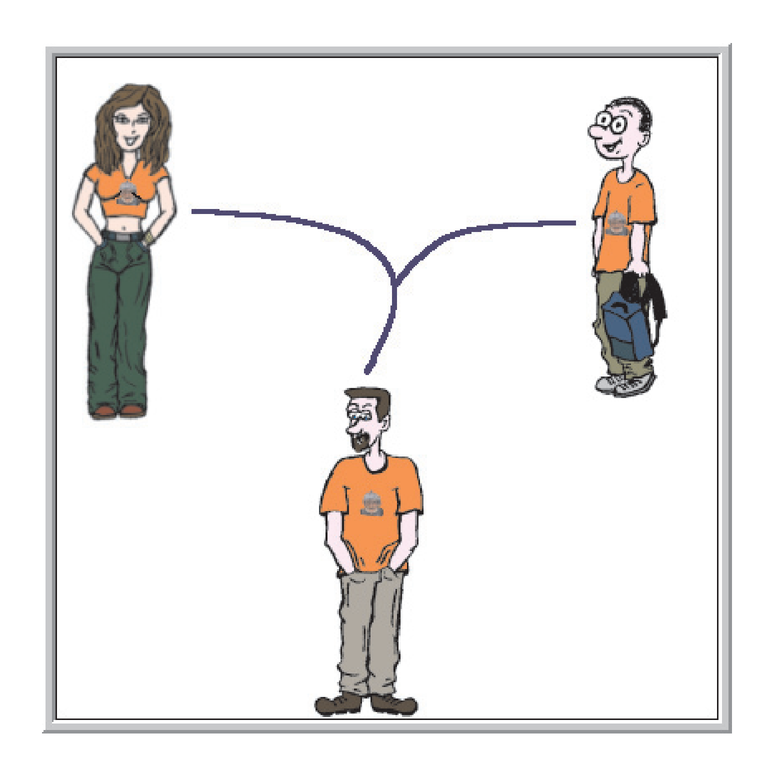
Si bien todo sigue conectado de antemano, las conexiones virtuales emulan el mecanismo por el cual, antes de comenzar una comunicación entre dos puntos, es necesario poner a ambas partes de acuerdo y establecer una conexión entre ambas.

En teoría, las conexiones virtuales permiten, igual que una conexión real, que la comunicación no sea escuchada por nadie que no esté conectado al "cable" que une ambos puntos, que terceras personas no puedan intervenir en la comunicación enviando datos que no sean de ninguna de las dos partes legítimas, que nadie pueda hacerse pasar por otro, etc., etc. **Por supuesto, todo esto es en teoría pero, como bien sabemos, nada en este mundo funciona como en teoría debería. ;-)**

¿Qué es lo que caracteriza una conexión?

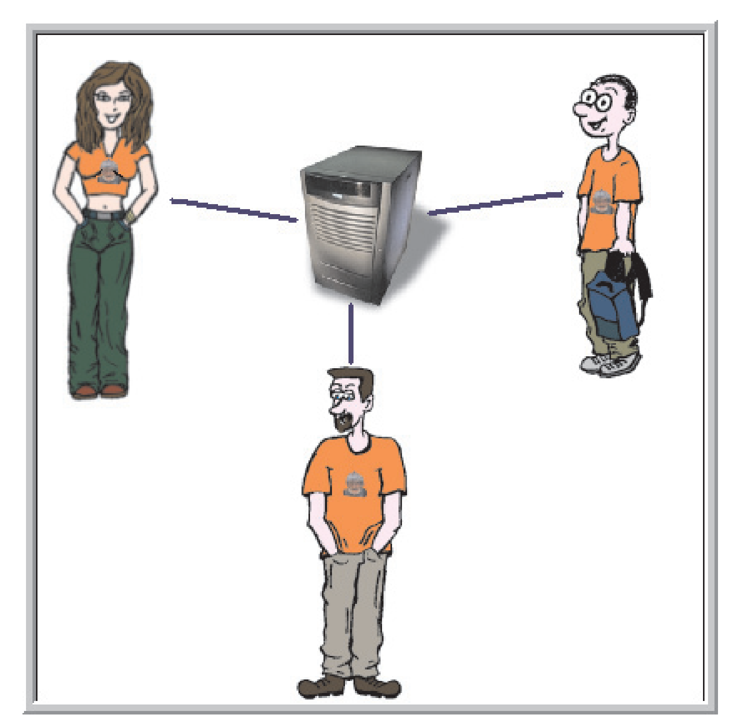
Las conexiones de las que hemos hablado son siempre **conexiones entre dos puntos**, y así son todas las conexiones en TCP. Aunque puedas estar en un chat hablando con 100 personas a la vez, en realidad toda esa comunicación se realiza a través de pares de conexiones.

Para entenderlo, veamos el caso de un chat entre 3 personas. Podríamos pensar en primer lugar en una solución que fuese un único cable con 3 conectores que conectase entre sí a los 3 interlocutores.



Pero, ¿que ocurriría si quisiéramos meter a una persona más en la conversación? ¿Tendríamos que quitar el cable de 3 conectores y cambiarlo por uno de 4? Sería mucho más sencillo un sistema en el cual todo funcionase mediante cables de sólo dos conectores. Para que alguien entrase o saliese del chat, bastaría con conectar o desconectar un sólo cable, pero no habría que modificar toda la red, y el resto de usuarios del chat no se verían afectados por lo que hiciera uno solo.

Para poder realizar este chat entre 3 personas mediante cables de 2 conectores se hace imprescindible la presencia de **un intermediario** que gestione todas las conexiones de todos los usuarios. Esta es precisamente la misión de un **servidor** de chat.



Como vemos en esta otra imagen, ahora hay 3 cables en lugar de uno sólo, y cada cable conecta a un único usuario con el servidor que gestiona el chat.

Es importante comprender que esto que hemos dicho de los cables de sólo dos conectores **no se refiere a las conexiones físicas de Internet**, si no a las conexiones virtuales que nos da el protocolo TCP.

De hecho, muchas redes TCP/IP tienen físicamente conectadas todas sus máquinas entre sí, en lugar de realizar las conexiones mediante cables punto a punto, pero insisto en que esto sólo ocurre con las conexiones físicas, pero no con las conexiones virtuales, que son siempre punto a punto.



Hablando claro...

Hablando claro, no importa como conectes **físicamente** tu PC a otros PCs (hay mil maneras de hacerlo, unas son punto a punto y otras no). Lo importante es que **TU PC** (y cualquier otro PC del mundo), **cuando utilice el protocolo TCP, establecerá siempre conexiones virtuales punto a punto**.

Por lo que hemos visto hasta ahora, es obvio que lo primero que caracteriza a una conexión son los dos extremos que conecta. En nuestro contexto, los extremos de la conexión se identifican mediante las conocidas **direcciones IP**.

Pero la identificación de los extremos no es lo único que caracteriza una conexión. Cada uno de los extremos podría tener varios "conectores" diferentes, así que no sólo basta saber con **quién** conectas, si no también **dónde**. En nuestro contexto, cada uno de los conectores de un mismo extremo serían los diferentes **servicios**: HTTP, FTP, POP3, etc. Por tanto, al igual que ocurría con UDP, otro de los parámetros que caracterizan una conexión TCP son los **puertos de conexión** de ambos extremos.

Si se tratase de conexiones físicas, no haría falta mucho más para ser caracterizadas. En cambio, al tratarse de conexiones virtuales, necesitamos algún mecanismo que identifique una conexión establecida. Y es aquí donde vemos la principal diferencia entre TCP y UDP.

Los paquetes TCP contienen un campo en su cabecera que no contienen los paquetes UDP. Este campo adicional es el que permite identificar la conexión virtual a través de la cual circula el paquete.

La solución más obvia que podríamos encontrar para implementar este identificador de conexión sería utilizar un número que identificase unívocamente la conexión. Pero esta solución presentaría varios problemas, el mayor de los cuales sería quizá la seguridad.

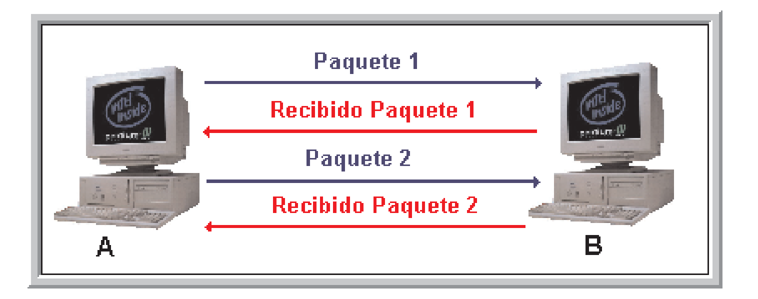
Si una conexión se identifica permanentemente por un mismo número, es sólo cuestión de tiempo que un atacante encuentre ese número por fuerza bruta y así, con sólo spoofear su IP (utilizar algún mecanismo para enviar paquetes en nombre de una IP que no sea la suya), se pueda colar en la conexión ajena. Una solución mucho mejor es utilizar un número que vaya cambiando con cada paquete, según unas ciertas "normas" establecidas entre los dos extremos.

Esto en realidad no es tal y como lo pinto. Este número cambiante realmente existe, pero su principal finalidad no es la de identificar conexiones, si no más bien la de mantener el estado de evolución de una conexión, y saber así en todo momento qué paquetes se pueden enviar y qué paquetes se pueden recibir a través de esa conexión.

Este número es el llamado **número de secuencia**, pero mejor será que no nos precipitemos más, que ya habrá tiempo de ver todo esto en detalle.

2.2. Fiabilidad en la comunicación

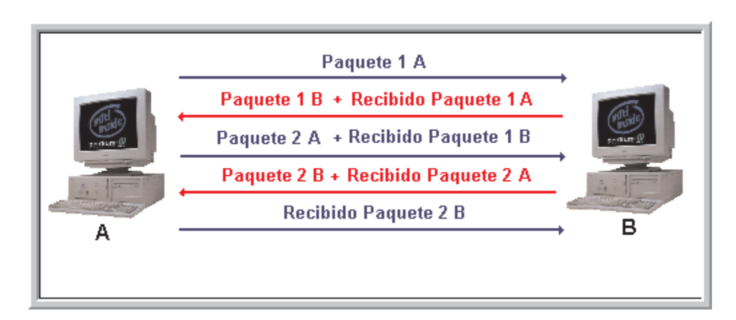
Como ya conté en el artículo sobre UDP, al contrario que éste, el protocolo TCP tiene un mecanismo para asegurar que los datos llegan correctamente a su destino. Para conseguir esto, cada paquete TCP que llegue a un destino ha de ser respondido por éste mediante otro pequeño paquete que confirme la recepción.



Esto sería así en el caso más sencillo, pero menos común, que sería el de una comunicación unidireccional, donde sólo una de las partes enviase datos a la otra.

En la práctica, la mayoría de los servicios son bidireccionales, por lo que es necesario intercalar los paquetes transmitidos con las confirmaciones de los paquetes recibidos.
Esto podría dar lugar a un gran número de paquetes (el doble, exactamente), ya que a cada paquete de datos habría que sumar además el correspondiente paquete de confirmación.

Una solución mucho más eficiente es englobar en un sólo paquete los datos transmitidos y la confirmación de los datos recibidos.



En esta figura vemos cómo las máquinas **A** y **B** intercambian paquetes a través de una misma conexión, actuando ambos simultáneamente como transmisores y receptores.

En primer lugar, la máquina **A** envía el primer paquete (Paquete 1 A). Una vez recibido el paquete 1 A, la máquina **B** decide también empezar a enviar sus propios paquetes, por lo que envía un único paquete a **A** en el que engloba la confirmación de que recibió el paquete 1 A, así como su propio paquete 1 B.

A esto responderá la máquina **A** enviando el siguiente paquete de los que desea transmitir, pero englobando en el mismo también la confirmación de que recibió correctamente el paquete 1 B.

Así continuaría la comunicación, hasta que la máquina **A** decidiese dejar de transmitir sus propios datos, por lo que su única misión consistiría ya sólo en seguir confirmando la recepción de los paquetes que le envía **B**, como ocurre en la figura con el caso del paquete 2 B, al cual la máquina **A** responde sólo con una confirmación de recepción, pero sin englobar más datos en el mismo paquete, ya que **A** no tiene ya nada más que desee enviar.

¿Qué ocurre si uno de los paquetes se pierde por el camino? Cada vez que una máquina transmite un paquete TCP, ésta no lo borra de su memoria, si no que lo deja almacenado aún por un tiempo, esperando que llegue la confirmación de su recepción. Si ésta llega, se podrá borrar el paquete de la memoria, y olvidarse para siempre de él. En cambio, si pasado un tiempo prudencial no ha llegado la confirmación de su recepción, se asumirá que el paquete no ha llegado a su destino, por lo que éste se retransmitirá. Así, el paquete se conservará en la memoria todo el tiempo que sea necesario, reenviándolo cada cierto tiempo, hasta que al fin llegue la confirmación de que el paquete fue recibido.

En el caso de que, debido a estos reenvíos, un mismo paquete llegue más de una vez al mismo destino, no habrá ningún problema, ya que el diseño de TCP permite diferenciar perfectamente un paquete de otro, por lo que al detectar la duplicación simplemente se rechazará la copia.

2.3. Orden en los datos

Tal y como decíamos también en el anterior artículo, TCP identifica cada paquete con un número de secuencia que permite ordenarlos correctamente independientemente de su orden de llegada al destino.

En el caso de UDP, los paquetes se iban procesando según iban llegando. En cambio, en TCP un paquete se puede dividir en fragmentos que podrán llegar en cualquier orden, y los datos no se procesarán hasta que se hayan recibido todos los fragmentos y estos se hayan ordenado según el número de secuencia de cada paquete.

2.4. Tratamiento de paquetes grandes

Esta capacidad de dividir los datos en fragmentos ordenados, combinada con el sistema de confirmación de respuestas de TCP, convierte a éste protocolo en un protocolo de transporte ideal para la transmisión segura de grandes cantidades de datos.

2.5. Control de flujo

Aquí entramos ya en un nuevo concepto que habrá que introducir, ya que es uno de los conceptos básicos del campo de las redes y telecomunicaciones.

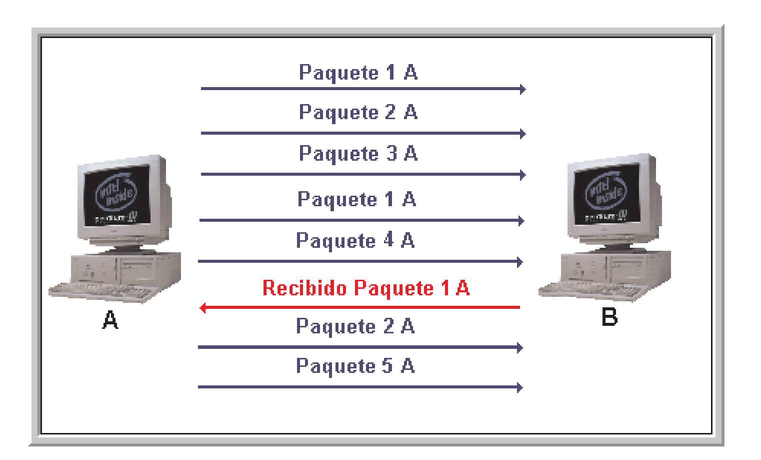
Hasta ahora hemos tratado las redes y las máquinas como si fuesen perfectas, sin limitaciones. Pero sabemos muy bien que en la vida real esto no es así ya que, por ejemplo, el **ancho de banda** de una red es uno de los factores más determinantes de su efectividad.

Hemos asumido que los paquetes se iban transmitiendo a diestro y siniestro, y que estos iban a ser recibidos sin problemas. Bueno, no exactamente... Hemos dicho que sí que podía haber problemas en la recepción, pero realmente no hemos hablado de los problemas del ancho de banda.

Si la única solución ante la congestión de la red fuese el mecanismo explicado anteriormente para asegurarnos de que todos los datos llegan al destino, el problema no se solucionaría si no que, al contrario, sería cada vez mayor.

Imaginemos la situación. La máquina **A**, que transmite los paquetes, es demasiado rápida para lo que es capaz de procesar

la máquina **B**, que recibe los paquetes. Por tanto, la máquina **A** empieza a enviar paquetes a toda velocidad y sin compasión, esperando las confirmaciones de la máquina **B**. Como **B** no da abasto, no es capaz de enviar las confirmaciones de recepción a tiempo. Al no llegar las confirmaciones, **A** se pondrá a reenviar los paquetes anteriores, al mismo tiempo que continúa enviando paquetes a diestro y siniestro, por lo que la congestión, sumando los paquetes nuevos y los viejos, será muchísimo mayor.



Si no se implementa algún mecanismo para evitar este caos, la situación puede hacerse insostenible en poco tiempo. Un mecanismo que se encargue de ajustar el flujo de la comunicación es precisamente lo que se conoce como **control de flujo**.

En el caso de TCP, disponemos de un sencillo mecanismo de control de flujo, que consiste en incluir en la cabecera de cada paquete un campo que indica al otro extremo de la comunicación cómo estamos de congestionados. Esta medida de la congestión la llevamos a cabo diciendo cuántos bytes estamos preparados para recibir. Según nos vayamos saturando, este número será cada vez menor, y el otro extremo de la comunicación, a no ser que sea un mal-nacido (que los hay), tendrá que actuar en consecuencia, relajándose un poco y dándonos tiempo a asimilar lo que ya nos ha enviado.

Más adelante veremos en detalle cómo se implementa el control de flujo en TCP.



Todo esto que...

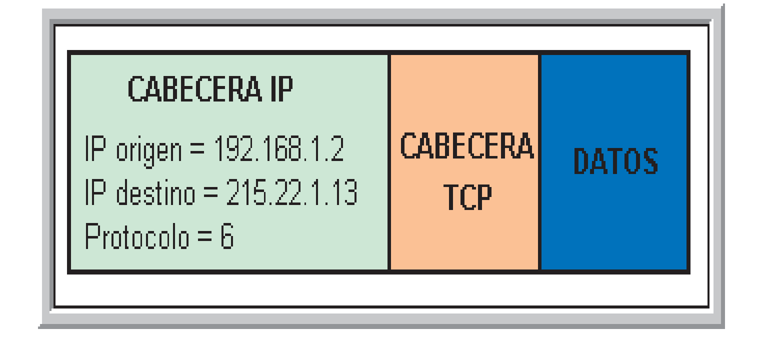
Todo esto que estamos explicando después lo "tocaremos" en la "realidad". Veremos los paquetes TCP y dónde está localizado cada concepto del que hablamos.

3. LO QUE SÍ QUE TIENEN EN COMÚN TCP Y UDP

Como no todo va a ser diferencias, vamos a ver los puntos en común entre ambos protocolos de transporte que, en el fondo, son bastante parecidos.

En primer lugar, TCP también se apoya sobre el protocolo **IP** y, en este caso, el **número de protocolo** asignado a TCP para que la capa de red (la capa IP) pueda identificar el protocolo de transporte, es el número **6**.

Como ya sabemos, la cabecera IP contendrá entre sus campos uno que identifique el protocolo de transporte que está por encima de él.



El punto en común más importante entre TCP y UDP es que el mecanismo de identificación de servicios es el mismo: cada paquete, tanto en TCP como en UDP, lleva dos **números de puerto** (origen, y destino), ambos comprendidos entre 0 y 65535 (16 bits).

Los puertos de TCP funcionan exactamente igual que los de UDP, pero no tienen ninguna relación entre sí, es decir, existen 65536 puertos para TCP, y otros 65536 puertos diferentes para UDP. Si, por ejemplo, en un firewall cierras el puerto 21 de TCP, no estarás cerrando al mismo tiempo el puerto 21 de UDP, ya que son totalmente independientes.

Otro punto en común es que ambos protocolos incluyen una **suma de comprobación** en cada paquete, que verifica la corrección de los datos incluidos en el mismo. La cabecera que se codifica en la suma de comprobación de TCP es muy similar a la que se utiliza en UDP.

4. LA CABECERA TCP EN DETALLE

Entramos ya en los detalles técnicos del protocolo TCP. En este caso, el RFC que los especifica es el **RFC 793**. Desde la página de los RFC (http://www.rfc-editor.org) podemos bajar tanto versión en **TXT** (http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=793&type=ftp&file_format=txt), como versión en **PDF** (http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=793&type=ftp&file_format=pdf).



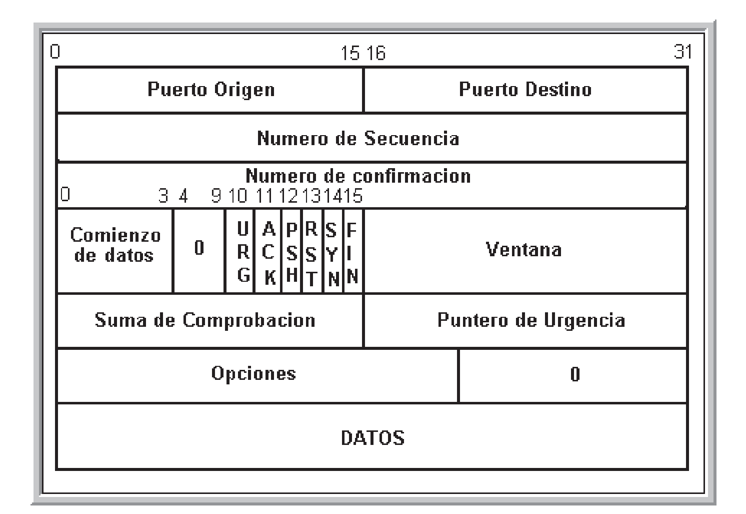
RFC 793 en español

RFC 793 en perfecto español!!! Como ya hemos comentado muchas veces, tenemos traducidos muchos de los RFC al español en la Web **www.rfc-es.org**. Este es el trabajo de muchas personas que desinteresadamente están colaborando en un proyecto realmente descomunal: traducir los más de 3000 documentos, si controlas de Ingles y te animas, ya sabes :)

El RFC 793 en castellano está en **http://www.rfc-es.org/getfile.php?rfc=0793**

Como vemos, el RFC tiene 91 páginas, por lo que es bastante más denso que la mayoría de los RFCs que hemos visto hasta ahora (tanto en la serie RAW como en el curso de TCP/IP). Aún así, no es demasiado duro de leer, sobre todo si prescindimos de algunas cosas poco importantes para la comprensión del protocolo, como la especificación de las variables del TCB, o toda la parte del final relativa al procesamiento de eventos. En cualquier caso, ya sabéis que aquí tendréis un buen resumen de todo lo más importante. ;-)

Como ya somos expertos en protocolos, vamos a plantar aquí directamente la cabecera TCP. **No pierdas de vista esta imagen**, haremos referencia a ella a lo largo de todo el artículo.



Bueno, bueno, la cosa ya va siendo más seria, ¿eh? ;-)
¡Que nadie se asuste! Ya veréis como en un momento comprendemos todo lo que hay en esa imagen aparentemente tan complicada.

Vamos a ir viendo campo por campo toda la cabecera, para luego entrar en detalle en el tamaño de los campos a nivel de bit (ya en el próximo artículo).

4.1. Puerto de origen

Este campo tiene el mismo significado que en el caso de UDP. En TCP es de especial importancia, porque es condición necesaria (aunque no suficiente) para identificar una conexión virtual.

4.2. Puerto de destino

También tiene el mismo significado que en UDP, y también es condición necesaria (aunque no suficiente) para identificar una conexión virtual.

Es decir, si nos llega un paquete que coincide en IP de origen, IP de destino, Puerto de origen, y número de secuencia, con los de una conexión virtual, pero no coincide el puerto de destino, entonces ese paquete no se puede considerar perteneciente a esa conexión virtual. Lo mismo ocurre si el que cambia es el puerto de origen.

4.3. Número de Secuencia

Aquí empieza lo más divertido, ya que este campo es uno de los que más nos dará que hablar. El número de secuencia es un número bastante grande (32 bits) que tiene varias funciones.

Por un lado, **identifica unívocamente a cada paquete** dentro de una conexión, y dentro de un margen de tiempo. Es este número el que nos permite detectar si un paquete nos llega duplicado. Durante un margen de tiempo (que depende del flujo de los datos y, por tanto, también del ancho de banda) tenemos la garantía de que en una determinada conexión virtual no habrá dos paquetes diferentes con el mismo número de secuencia.

Por ejemplo, en una conexión de 2Mbps, este margen de tiempo es de unas 4 horas y media, mientras que en una conexión de 100Mbps el margen es de 5’4 minutos.

¿Por qué este margen de tiempo varía en función del flujo de datos? Pues lo comprenderemos en cuanto veamos qué es exactamente el número de secuencia.

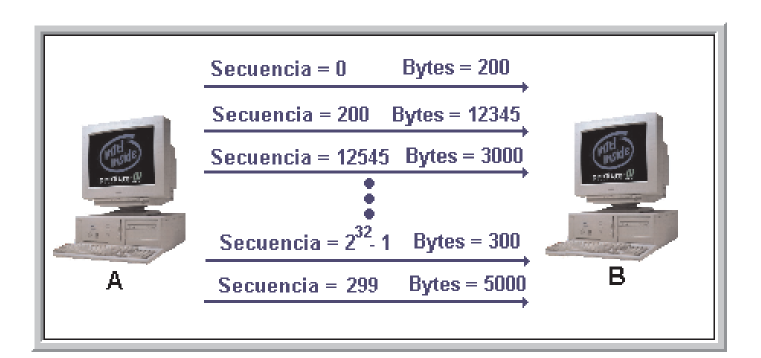
Y es que el número de secuencia es mucho más que un mero identificador de paquetes. El valor del número de secuencia no es aleatorio, si no que tiene un significado muy concreto. El número de secuencia es el **orden en bytes que ocupan los datos contenidos en el paquete, dentro de una sesión**.

Es decir, supongamos que el primer paquete enviado una vez establecida una conexión tiene un número de secuencia 0, y el paquete contiene 200 bytes de datos. El siguiente paquete que se enviase tendría que tener 200 como número de secuencia, ya que el primer paquete contenía los bytes del 0 al 199 y, por tanto, el orden que ocupan los datos del segundo paquete es el siguiente: 200.

¿Por qué decíamos entonces que el número de secuencia identifica unívocamente un paquete dentro de un margen de tiempo? Porque el número de

secuencia es siempre creciente, por lo que un paquete posterior a otro siempre tendrá un número de secuencia mayor. Claro que... será mayor en módulo 32. Esto significa que, una vez que el número de secuencia valga $2^{32} - 1$, es decir, el mayor número que se puede representar con 32 bits, el próximo número de secuencia utilizado será 0, es decir, se dará la **vuelta al marcador**.

Por supuesto, cuánto se tarda en dar la vuelta al marcador depende directamente de cuántos datos se envíen, y en cuanto tiempo.



En la imagen podemos ver reflejado esto. Como vemos, el primer número de secuencia es 0, y el primer paquete contiene 200 bytes de datos. Por tanto, el segundo paquete tendrá número de secuencia 200. Este paquete contiene 12345 bytes de datos, por lo que el próximo paquete tendrá como número de secuencia: 200 (número de secuencia anterior) + 12345 = 15345.

Así continúa la sesión, hasta que un paquete tiene como número de secuencia $2^{32} - 1$, y contiene 300 bytes de datos. En teoría, el próximo número de secuencia tendría que ser $2^{32} - 1 + 300$ pero, como solo contamos con 32 bits para representar este número, tendremos que dar la vuelta al marcador, es decir, convertir el 2^{32} en un 0, por lo que nos quedaría: 0 –1 + 300 = 299. Por tanto, 299 sería el número de secuencia del próximo paquete.

En realidad, el número de secuencia no tiene por qué comenzar en cero. El principal motivo es que si todas las conexiones empezasen en 0, no se podrían **reutilizar las conexiones**.

Ya a estas alturas, habiendo visto todos los campos relevantes, podemos afirmar que lo que identifica una conexión son estos 5 parámetros**: ip de origen, ip de destino, puerto de origen, puerto de destino, y número de secuencia**.

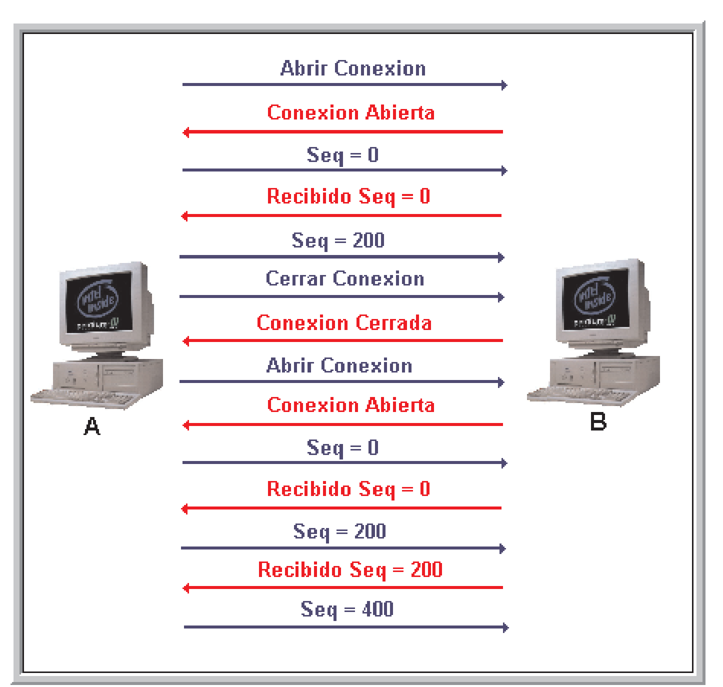
Imaginemos que, por ejemplo, nuestro software de correo electrónico está configurado para comprobar si tenemos mensajes nuevos cada 30 segundos. El programa tendrá que establecer una conexión TCP/IP con el puerto 110 (puerto de POP3) de un servidor POP3 cada 5 segundos. En este caso, 4 de los parámetros que identifican la conexión podrían ser iguales: puerto de origen, puerto de destino, ip de origen, e ip de destino.

Digo que "podrían" porque el puerto de origen se podría cambiar de una conexión a otra (el resto de parámetros no se podrían cambiar de ninguna manera), aunque esto en muchos casos no será posible o conveniente. Por tanto, el único parámetro con el que podemos jugar para poder establecer nuevas conexiones similares sin que se confunda la nueva con una vieja que ya se cerró, es el número de secuencia.

Si cada vez que el programa de correo se reconecta al servidor POP3 utilizamos el mismo número de secuencia para comenzar la sesión (por ejemplo, 0), si

llegase un paquete muy retrasado del servidor de correo de una conexión anterior (de los 5 segundos anteriores), no habría forma de saber si este paquete pertenecía a la sesión anterior, o a la nueva sesión. Una forma sencilla de evitar esto es no utilizar siempre el mismo número de secuencia para comenzar una sesión, si no ir utilizando números de secuencia cada vez mayores.

Veamos un ejemplo de esto a partir de la siguiente figura.



En primer lugar, la máquina **A** abre una conexión con la máquina **B**. Una vez abierta la conexión, **A** empieza a transmitir sus datos, empezando por un paquete con número de secuencia Seq = 0. **B** lo recibe y devuelve su confirmación de recepción.

A continuación, **A** transmite un nuevo paquete, con número de secuencia 200 (por tanto, el primer paquete tenía que contener 200 bytes de datos). Por algún azar del destino, este paquete va a tardar más de lo normal en llegar a **B**. Como **A** ya ha enviado lo que quería, cierra la conexión, a pesar de que todavía no ha recibido la confirmación de que **B** recibió el paquete Seq = 200.

Inmediatamente después, **A** decide establecer una nueva conexión con **B** para enviarle otros datos. Comienza enviando un primer paquete con Seq = 0, al cual responde **B** con su confirmación de recepción. Para entonces, ya ha llegado a **B** el paquete Seq = 200 de la anterior conexión, por lo que construye su paquete de confirmación. Pero la mala suerte se confabula contra estas dos máquinas, y justo en ese instante **A** envía un nuevo paquete, que tiene también número de secuencia 200. Inmediatamente **B** envía la confirmación de recepción del paquete Seq = 200 de la anterior conexión.

¿Qué ocurre a partir de aquí?

1.- Desde el punto de vista de **A**, **B** ha recibido correctamente el último paquete, por lo que continúa transmitiendo nuevos paquetes, borrando de su memoria el paquete Seq = 200, por lo que nunca lo reenviará.

2.- Desde el punto de vista de **B**, acaba de recibir un nuevo paquete Seq = 200. Como **B** ya había recibido un paquete Seq = 200, interpreta que éste paquete es un reenvío del anterior, por lo que lo desecha.

Por tanto, ni **B** recoge el paquete Seq = 200, ni A se entera de lo que ha pasado. En realidad las cosas no son exactamente como las pinto, pero ya iremos viendo más adelante cómo es todo esto realmente.

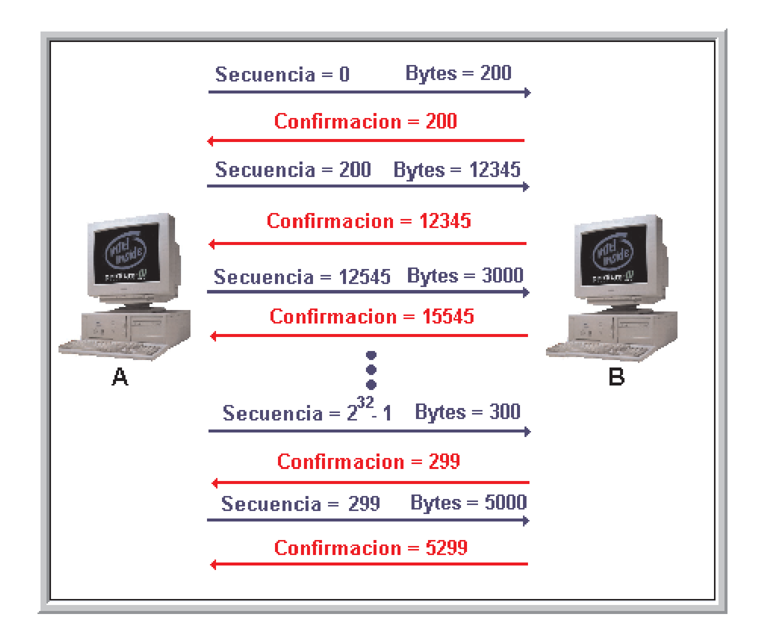
Por supuesto, el número de secuencia no sirve sólo para detectar paquetes duplicados, o para diferenciar unas conexiones de otras, si no también para poder ordenar los paquetes. Como el número de secuencia representa el orden exacto en bytes de los datos, es imposible confundirse a la hora de unir los datos en el orden correcto.

4.4. Número de confirmación

Este campo es el que se utiliza para hacer las confirmaciones de recepción de las que tanto hemos hablado. Su funcionamiento es muy similar al del número de secuencia, porque en realidad también representa un **orden en bytes de los datos**.

Lo que se envía en el número de confirmación es otro número de 32 bytes que indica **cuál es el próximo byte que estamos esperando recibir**.

Si nuestro compañero en una conexión nos envía un paquete con número de secuencia 2000, y el paquete contiene 350 bytes de datos, el próximo número de secuencia sería el 2350. Si queremos confirmar que hemos recibido el paquete con número de secuencia 2000, tendremos que enviar en nuestro paquete de respuesta un número de confirmación que valga 2350, diciendo así*: "Ya puedes enviarme el paquete 2350. He recibido todo bien hasta aquí"*. Vamos a ver la misma figura que pusimos al hablar sobre el número de secuencia, pero esta vez incluyendo los números de confirmación.



Este campo sólo tiene significado si el paquete tiene activo el **flag ACK**, pero eso ya lo veremos más adelante.

4.5. Comienzo de datos

Este campo no era necesario en el caso de UDP, ya que la cabecera UDP tiene un tamaño fijo de 8 bytes. En cambio, la cabecera TCP puede tener un tamaño variable, tal y como veremos más adelante, por lo que es necesario indicar a partir de qué punto comienzan los datos y termina la cabecera.

Este punto de comienzo no se expresa en bytes, como podríamos esperar, si no en **palabras de 32 bits**. Si nos fijamos en la cabecera, veremos que está estructurada en filas, cada una de las cuales mide 32 bits. La primera fila contiene los puertos de origen y destino, la segunda el número de secuencia, la tercera el número de confirmación, etc.

El valor más habitual para este campo es **5**, que equivale a **20 bytes de cabecera**, es decir, lo mínimo que puede ocupar la cabecera TCP.

4.6. Espacio reservado

Después del campo comienzo de datos, vemos que hay uno en el que simplemente pone 0. Este es un espacio reservado de 4 bits, que hay que dejar a cero, sin más.

4.7. FLAGS (URG, ACK, PSH, RST, SYN, FIN)

Aquí tenemos también un asunto que va a dar mucho que hablar. Los flags son una serie de indicadores de control, de un único bit cada uno, con diferentes funciones que detallo a continuación.

4.7.1. Flag URG (Urgent)

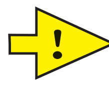
Cuando este flag está activo (vale 1, en lugar de 0), estamos indicando que el paquete contiene **datos urgentes**. La especificación de TCP no detalla qué se debe hacer exactamente ante unos datos urgentes, pero lo normal es atenderlos con máxima prioridad.

Si, por ejemplo, estamos procesando datos anteriores, y nos llega un paquete con datos urgentes, normalmente aplazaremos el procesamiento de los datos anteriores para prestar toda nuestra atención a los datos urgentes.

¿Y para qué pueden servir los datos urgentes? No es habitual encontrarlos, pero un claro ejemplo es cuando deseamos abortar alguna acción. Por ejemplo, si alguna vez habéis conectado por **Telnet** con un sistema remoto (el propio servicio de Telnet del puerto 23, no los experimentos raros que hacíamos en la serie RAW ;-) sabréis que con la combinación de teclas **CONTROL- C** se puede abortar cualquier acción.

Imaginemos que queremos ver el contenido de un archivo remoto, por ejemplo en Linux, y hacemos un **cat** del archivo:

cat archivo.txt



¿Qué no sabéis...

¿Qué no sabéis lo que es cat? Pues tendréis que repasar los artículos ya publicados sobre Linux de esta misma revista. ;-)... y si no, consulta el google (www.google.com).

El caso es que, después de hacer el cat, descubrimos que el archivo es mucho más largo de lo que esperábamos y, para colmo, no nos interesa. Pulsamos **CONTROL-C** y el listado del archivo se aborta sin necesidad de esperar a que el archivo termine de mostrarse entero.

El paquete que hemos enviado a la máquina remota conteniendo la orden de abortar el listado, tiene un flag **URG** activo.
Si no existiese este mecanismo, probablemente la máquina remota seguiría enfrascada en su asunto inmediato, que es mostrar el listado del archivo, y no habría hecho caso a nuestro **CONTROL-C** hasta que no hubiera terminado con lo suyo.

Para garantizar una mayor efectividad del flag **URG** hay que combinarlo con el flag **PSH**, que veremos más adelante.

4.7.2. Flag ACK (Acknowledge)

Cuando este flag está activo (vale 1, en lugar de 0) significa que nuestro paquete, aparte de los datos propios que pueda contener, contiene además una

confirmación de respuesta a los paquetes que está enviando el otro extremo de la conexión.

Por tanto, el campo **número de confirmación** no tiene significado a no ser que esté activo este flag.

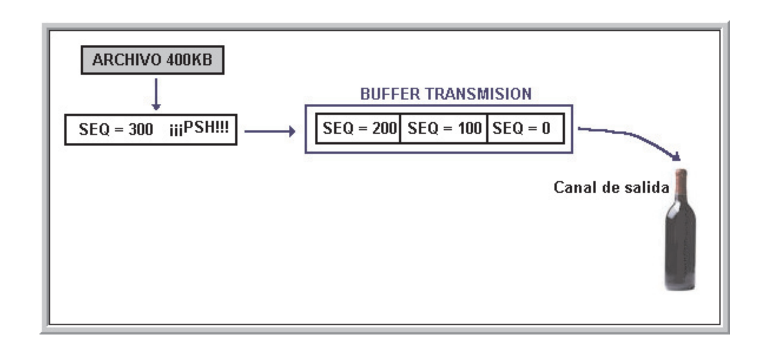
4.7.3. Flag PSH (Push)

Cuando este flag está activo (vale 1, en lugar de 0) indicamos a nuestro propio sistema, y al sistema remoto con el que estamos conectados, que deben vaciar sus buffers de transmisión y recepción... Creo que ha llegado el momento de hablar sobre los buffers de transmisión y recepción. :-m

En todo sistema TCP tiene que haber dos pequeñas memorias, o **buffers**: una para transmisión, y otra para recepción. Estas memorias son unas colas de paquetes en las que se van almacenando los paquetes que hay que procesar en espera del momento adecuado.

Por ejemplo, supongamos que una aplicación que corre en nuestro sistema desea enviar un archivo, que será partido en 4 paquetes TCP. A pesar de que nuestra aplicación lance la orden de enviar el archivo en un sólo instante, nuestro sistema no podrá enviar de golpe los 4 paquetes. Por tanto, tendrá que construir los paquetes, y luego ponerlos en una cola para que se vayan enviando uno tras otro.

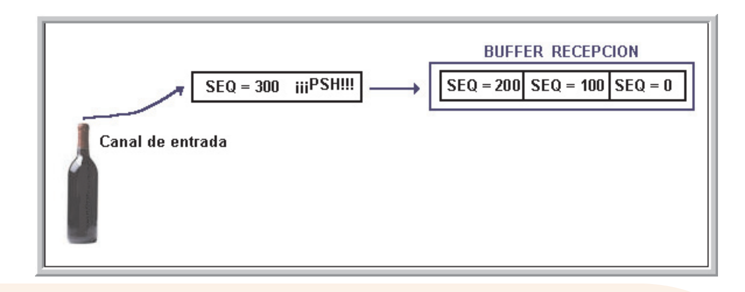
Si el sistema pudiera enviar los 4 paquetes a la vez no habría necesidad de tener ninguna cola, pero en la práctica, como en cualquier cola, sólo se puede pasar de uno en uno.



En la imagen podemos ver cómo los paquetes sin flag **PSH** se van encolando en el **buffer de transmisión**. Cuando el sistema construye el último paquete (el paquete Seq = 300), ya puede comenzar el envío del archivo. Por tanto, en el último paquete incluye el flag **PSH**.

A partir del instante en que hemos metido en el buffer el paquete con flag **PSH**, el sistema irá enviando uno tras otro los paquetes a través del cuello de botella que supone el canal de salida. Se trata de un cuello de botella porque por él sólo puede pasar un único paquete en cada instante, por lo que los paquetes tendrán que ir enviándose de uno en uno según el orden en que fueron ubicados en el buffer de transmisión. Al tratarse de una cola, el primero en llegar será el primero en salir (lo que se llama una cola **FIFO**), por lo que primero saldrá el paquete con Seq = 0.

El último paquete en salir será el que tiene Seq = 300, que es precisamente el que lleva el flag **PSH**. En la siguiente imagen vemos todo esto desde el punto de vista del receptor.



Por el canal de entrada irán llegando los paquetes uno tras otro. Se irán almacenando en el buffer de recepción, y no serán procesados hasta que llegue el último paquete, que contiene el flag **PSH**. Como este paquete es el que completa el archivo de 400B, para cuando llegue, el sistema receptor ya podrá procesar los datos de los 4 paquetes y reconstruir el archivo original.

Como decíamos, el flag **PSH** debe combinarse con el flag **URG** cuando haya datos urgentes. El flag **URG** será el encargado de decir al receptor que ha de atender los datos con máxima prioridad, y el flag **PSH** se asegurará de que el paquete no se retrase esperando en los buffers.

4.7.4. Flag RST (Reset)

Cuando este flag está activo (vale 1, en lugar de 0), estamos indicando al otro extremo de la conexión que algo no anda bien, ya que los datos que nos han llegado no coinciden con nuestra conexión, por lo que **se ha perdido la sincronización** entre ambas partes.

Ante cualquier campo incorrecto que recibamos (números de secuencia inválidos, o flags no esperados) tendremos que responder con un paquete con este flag activo, para que el otro extremo se entere del problema, y se cierre la conexión para re-sincronizar ambas partes.

Un uso típico de este flag es cuando estamos intentando conectar con un servidor, enviando varios paquetes para establecer la conexión, y al final uno de ellos tiene éxito. Si después de ese paquete de conexión siguen llegando al servidor el resto de nuestros intentos de conexión, el servidor nos responderá con **RST** a cada nuevo intento de conexión, para que nos olvidemos de esa conexión y nos centremos en la que ya tenemos.

4.7.5. Flag SYN (Synchronization)

Cuando este flag está activo (vale 1, en lugar de 0), estamos indicando al otro extremo que deseamos establecer una **nueva conexión**. Este flag se utiliza únicamente al abrir una nueva conexión.

Más adelante veremos el mecanismo exacto por el que se establecen las conexiones, así como algunas cuestiones de seguridad relacionadas con este flag.

4.7.6. Flag FIN (Finish)

Cuando este flag está activo (vale 1, en lugar de 0), indicamos al otro extremo de la conexión de que, por lo que a nosotros respecta, **la conexión ya se puede cerrar**.

Normalmente, una vez que enviemos nuestro propio **FIN**, tendremos que esperar a que nuestro compañero nos envíe el suyo. Una vez puestos los dos de acuerdo en que no hay más que hablar, se puede cerrar la conexión pacíficamente.

Tanto **RST** como **FIN** se utilizan para finalizar conexiones, pero la diferencia es que **RST** avisa de una situación de error, mientras que **FIN** avisa de una terminación sin problemas.

Ante una terminación con **RST**, normalmente las aplicaciones avisarán al usuario, por ejemplo con una ventana avisando que se ha perdido la conexión.

4.8. Ventana

Este campo es el que se utiliza para llevar a cabo el **control de flujo** implementado en TCP.

Como ya dije, el control de flujo permite evitar la congestión debida a la diferencia de velocidad (bien por capacidad de procesamiento, o bien por ancho de banda) entre ambas partes de una conexión.

Una forma muy sencilla de conseguir esto es hacer que cada extremo de la conexión vaya indicando a cada momento cómo de congestionado está. La mejor forma de indicar esta medida de congestión es decir cuántos bytes vamos a ser capaces de procesar en un momento dado.

Por tanto, a cada paquete le acompaña un número de 16 bits, que es el **número de bytes que estamos preparados para recibir** en el próximo paquete. Esto significa que, como máximo, un paquete podrá contener 65536 bytes de datos, es decir, 64KB, ya que es el máximo que podemos solicitar en cada momento.

Normalmente, el tamaño de la ventana está relacionado con la cantidad de espacio libre que tenemos en el **buffer de recepción**.

Vamos a ver un ejemplo.

Seq=0, Bytes=1000
Ack=1000, Window=1000
Seq=1000, Bytes=1000
Ack=2000, Window=500
Seq=2000, Bytes=1000
Ack=2500, Window=250
Seq=2500, Bytes=250
A
B

En primer lugar, la máquina **A** envía un paquete de 1000 Bytes, con número de secuencia 0.

La máquina **B** lo recibe y procesa correctamente, y envía su confirmación de recepción, indicando que la máquina **A** puede continuar enviando a partir del byte 1000 (Ack=1000). Además, le indica que está preparada para recibir otros 1000 bytes más. Si hacemos cuentas, en realidad lo que está diciendo la máquina **B** es que está preparada para recibir los bytes del 1000 al 1999.

La máquina **A** continúa transmitiendo, y lo hace esta vez con 1000 bytes más de datos, empezando, por supuesto, por el número de secuencia 1000.

La máquina **B** también los recibe sin problemas, pero se está empezando a agobiar un poco, por lo que avisa en su confirmación de que el próximo paquete no podrá ser tan grande y, como mucho, tendrá que ser de 500 bytes nada más.

La máquina **A**, en cambio, no ha recibido a tiempo la nueva ventana, y ha seguido enviando los paquetes al ritmo que llevaba anteriormente. Por tanto, el próximo paquete (Seq=2000) contiene también 1000 bytes de datos, a pesar de que la nueva ventana de **B** admite sólo 500 bytes.

Justo después de enviar este paquete demasiado grande, la máquina **A** recibe ya el aviso de **B** de que se está congestionando, por lo que decide esperar a ver cómo se las apaña su compañero.

Cuando **B** ha podido procesar unos cuantos datos, avisa a **A** de que puede seguir enviando, aunque sólo ha procesado 500 bytes de los 1000 que le envió y,

además, el próximo paquete tendrá que ser como máximo de 250 bytes.

En respuesta, **A** envía un nuevo paquete, comenzando en el byte 2500, y conteniendo tan sólo 250 bytes.

Todo esto es muy bonito, **A** y **B** son muy amigos, y **A** le va enviando a **B** las cosas poco a poco cuando éste se agobia. Pero esto en realidad no es una idea muy buena. Si **A** va haciendo caso a **B** según éste le vaya diciendo lo que puede manejar, y **B** va avisando a **A** según vaya liberando hueco en su buffer de recepción, los paquetes que se transmitan irán siendo cada vez más pequeños.

En el peor de los casos, podría llegar a ser 0 la ventana de **B**, y en el instante en que hiciese hueco para un mísero byte, avisar a **A**, por lo que **A** le enviaría un paquete con un único byte de datos. Esto daría lugar a una transferencia muy poco efectiva de los datos, donde muchos de los paquetes serían de un tamaño ridículo.

Por tanto, es aconsejable no ir ajustándose literalmente al tamaño de la ventana, si no dejar unos pequeños márgenes. Por ejemplo, cuando se detecte que la ventana va decreciendo por momentos, lo mejor es esperar un tiempo para dejar que la ventana vuelva a recuperar toda su capacidad, y continuar entonces la transmisión. Si fuésemos con prisas, empeñándonos en enviar más y más datos, la reducción de tamaño de la ventana sería cada vez mayor, y la transferencia cada vez menos eficiente.

En el caso de que la ventana llegue a ser cero, la responsabilidad del receptor (el que se ha congestionado) es avisar cuando su ventana se recupere, con un mensaje (vacío si hace falta) cuya única finalidad sea precisamente avisar del cambio de la ventana. Por otra parte, la responsabilidad del emisor (el que ha congestionado al otro) es dejar de enviar más datos, y esperar un tiempo prudencial para continuar la transmisión pasados unos dos minutos, si es que el receptor no ha enviado una nueva ventana antes.

4.9. Suma de comprobación

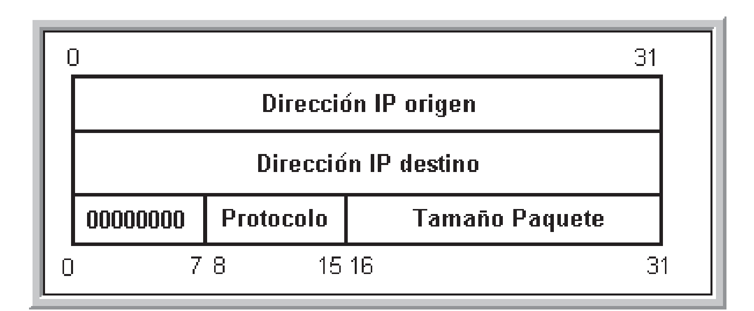
Al igual que en el caso de UDP, la suma de comprobación se calcula mediante una operación de aritmética binaria (suma en complemento a uno de 16 bits) que se realiza sobre una cabecera especial que contiene los datos más relevantes.

Cada vez que se recibe un paquete TCP, hay que realizar esta misma operación con los datos recibidos, y comparar el número obtenido con el campo **suma de comprobación** del paquete. Si ambos números no son iguales, los datos son incorrectos, y será necesaria una retransmisión de los mismos. En ese caso, no enviaremos la confirmación de recepción pertinente, esperando a que el emisor decida retransmitir el paquete al ver que no les respondemos.

Aunque ya lo comenté en el anterior artículo, os recuerdo que tenéis detallada la operación de la suma de comprobación en el **RFC 1071**, y que tenéis un código de ejemplo en C en la siguiente URL: http://www.netfor2.com/tcpsum.htm

En este caso no tenéis que realizar ninguna modificación sobre el código, ya que sirve precisamente para calcular la suma de comprobación de TCP.

La cabecera que se forma para llevar a cabo la suma de comprobación es la siguiente:



Como vemos, es igual que la de UDP, sólo que en este caso el **protocolo** es el **6**, en lugar del 17.

4.10. Puntero de urgencia

Cuando hablamos sobre el flag **URG** dijimos que servía para indicar que el paquete contiene datos urgentes.

Pero, ¿significa esto que los datos tienen que ir en un único paquete para ellos solos? Esto podría ser poco eficiente, ya que los datos urgentes podrían ser tan sólo unos pocos bytes, y nos obligaría a enviar paquetes casi vacíos sólo para poder transmitir el dato urgente.

Por ejemplo, el **CONTROL-C** del que hablábamos, es un comando muy breve, y sería un desperdicio enviar paquetes vacíos que sólo llevasen ese comando. Por tanto, **TCP permite combinar en un mismo paquete datos urgentes con datos no urgentes.**

Para conseguir esto, el campo puntero de urgencia nos indica el **punto a partir del cual terminan los datos urgentes**.

Si, por ejemplo, nuestro paquete contiene 1000 bytes de datos, y el puntero de urgencia es 150, sabremos que los 150 primeros bytes del paquete son urgentes, y los otros 850 son datos normales. Por supuesto, esto sólo tendrá sentido si el flag **URG** está activo. Si no es así, el campo **puntero de urgencia** simplemente será ignorado.



4.11. Opciones

Llegamos al fin al último campo de la cabecera TCP. Este campo es opcional, y es el responsable de que la cabecera TCP sea de **tamaño variable** y, por tanto, de la necesidad del campo **comienzo de datos**. En realidad, sólo existe una opción definida por el estándar en el **RFC** de TCP, pero la cabecera se diseñó con previsión de incluir otras opciones.

La opción definida se utiliza únicamente al establecer una nueva conexión. Lo que indica este campo opcional es el **máximo tamaño de los segmentos** que estamos dispuestos a recibir. Un segmento es cada una de las partes en las que se dividen los datos, por lo que nuestro compañero de conexión no debe enviarnos nunca paquetes más grandes de lo que indiquemos con esta opción. En el caso de que no usemos este campo opcional, nos podrá enviar paquetes de cualquier tamaño ajustándose, eso sí, a nuestra ventana de recepción.

¿Y qué relación hay entonces entre la ventana de recepción y el tamaño máximo de segmento?

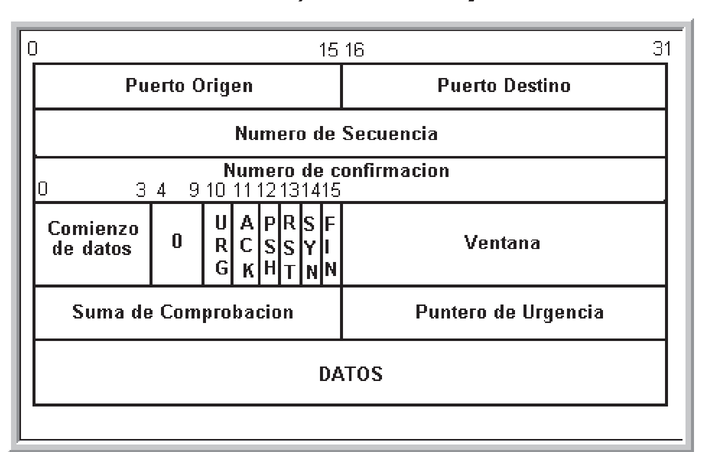
Pues la ventana es un parámetro dinámico, que va variando según la congestión que hay en cada momento, mientras que el tamaño máximo de segmento es una constante para toda la sesión, que se establece de antemano. Muchas veces, el tamaño máximo de segmento será menor que la ventana. Esto puede parecer absurdo, pero no es así, ya que el tener

una ventana mayor que el tamaño máximo de segmento indica que tenemos recursos suficientes para recibir varios paquetes en poco tiempo.

Por tanto, el tamaño máximo de segmento no es útil para realizar un control de flujo, si no tan sólo una restricción para el tamaño máximo que pueden tener los paquetes.

Pero vamos a ver cómo se indica exactamente esta opción. Como ya hemos visto, existen dos casos: o que no indiquemos **ninguna opción**, o que indiquemos el **tamaño máximo de segmento**.

En el primer caso, simplemente terminará en este punto la cabecera TCP, y comenzarán a partir de aquí los datos. Por eso, cuando hablábamos del campo **comienzo de datos** dijimos que el tamaño mínimo de la cabecera (que es este caso) es de **20 bytes**.



En el segundo caso, en cambio, habrá que rellenar el campo opciones de la cabecera. Y no sólo eso, si no que además hay que **ajustarlo a un tamaño de 32 bits**, que es lo que ocupa cada fila de la cabecera TCP (en realidad, se trata del tamaño de la **palabra**, ya que la idea de filas es sólo una abstracción para ver más fácilmente la estructura de la cabecera).

La opción que hemos mencionado, la única definida en el RFC, encaja exactamente en 32 bits, por lo que no hay que hacer ningún ajuste. El formato exacto de esta opción lo veremos más adelante cuando veamos las cabeceras a nivel de bits.

En cambio, no sólo existen las opciones definidas en el **RFC 793**. Podemos ver una lista de opciones TCP mantenidas, para variar, por el **IANA**, en: http://www.iana.org/assignments/tcp-parameters

Una opción que nos podremos encontrar con facilidad de las de esa lista, es la opción **SACK**, utilizada para confirmaciones selectivas (**S**elective **ACK**nowledgment).
Los detalles sobre esta opción los tenéis en el **RFC 2018** (ftp://ftp.rfc-editor.org/in-notes/rfc2018.txt).

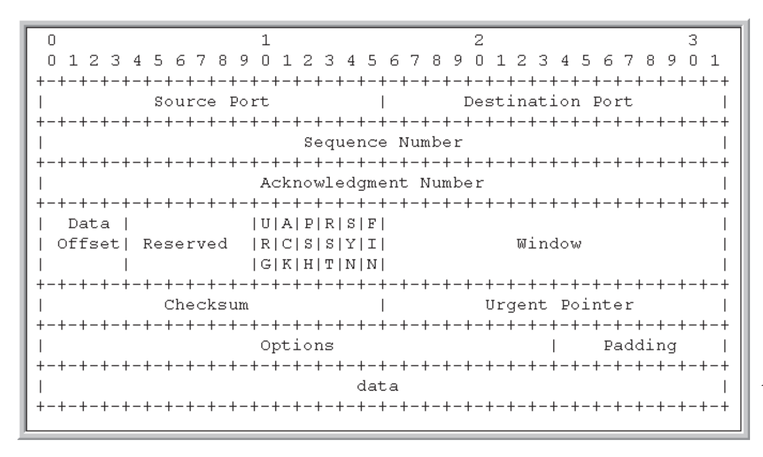
Si las opciones no encajan en 32 bits, habrá que completar la fila con ceros.

Para separar las opciones entre sí se utiliza una opción especial sin ningún efecto (**NOP** = **N**o **OP**eration) que ocupa un único byte.

Para terminar la lista de opciones existe también otra opción especial, de un único byte, que indica que no hay mas opciones y, por tanto, una vez completada la palabra de 32 bits, vendrán a continuación los datos del paquete.

Todo esto lo veremos en más detalle en el próximo artículo, en el cual empezaremos construyendo como ejercicio una cabecera TCP desde cero a bajo nivel, es decir, en binario puro y duro. Espero que estéis preparados! ;-)

Para terminar, os incluyo aquí la cabecera TCP con los nombres originales, en inglés, por si completáis este curso con alguna otra lectura, para que veáis la correspondencia y no os liéis.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



IMPORTANTE

IMPORTANTE:
Ya habrá tiempo de pasar a la practica, ahora es importante que empecemos a familiarizarnos con un montón de términos y conceptos que seguramente (para la mayoría) son completamente desconocidos.

No te preocupes demasiado si muchas cosas te han quedado "oscuras", ya empezarás a verlas claras cuando relacionemos la teoría con la practica y veas la utilidad "real" de cada concepto explicado aquí.

Autor: PyC (LCo)

XBOX LIFE VI
MENU X
POR ALFONSO MENKEL

- Descubriremos cómo realizar nuestros propios menus gracias a una versión hackeada de la version oficial.
- Meteremos varios Juegos en un DVD

Bienvenidos un mes más a la serie Xbox Life. Este mes vamos a ver el Menu X, que nos permitirá crear menús a nuestro gusto y así meter varios juegos en un mismo DVD.

Pues manos a la obra, aquí tenéis la lista de lo que necesitamos:

- Xbox con ModChip instalado.
- Grabadora de DvD.
- MenuX
- Nero 6.019 o Superior
- Editor de fotos (es opcional).

Todo menos el Menú X se ha explicado anteriormente, si no sabéis de lo que estoy hablando pasaros por la Web y pedid los números atrasados.

El software Menu X puedes descargarlo desde aquí en formato RAR:

http://www.megagames.com/console/cgi-bin/dl.cgi?id=xbox&file=cpxmenux!rar



También puedes...

También puedes descargarlo en formato ZIP de la Web de la revista (www.hackxcrack.com).

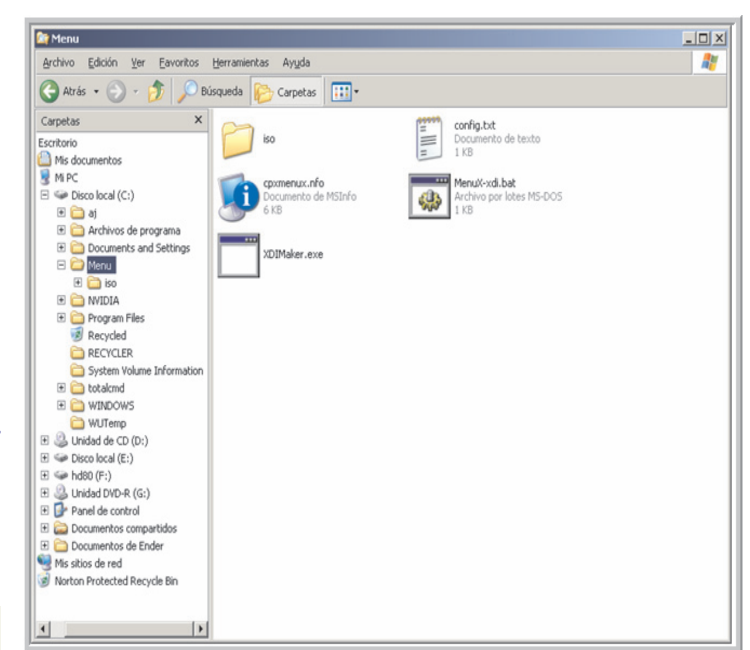


Este menú...

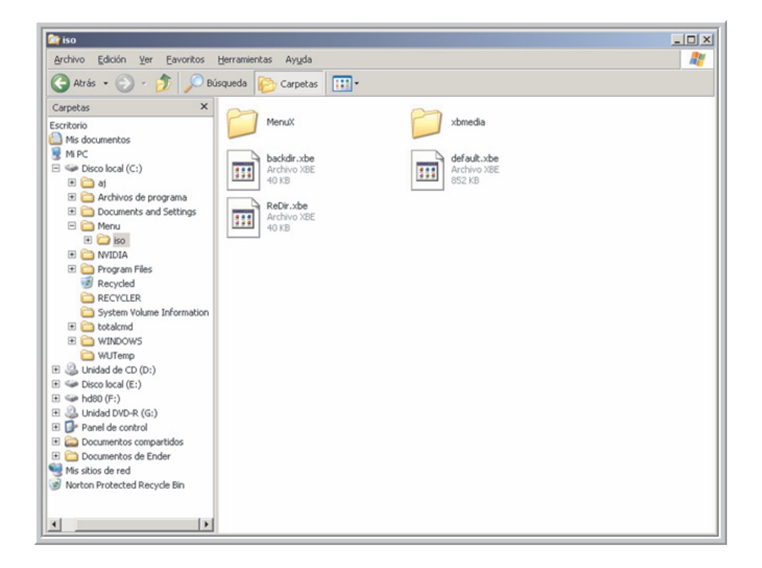
Este menú es el mismo que viene en las demos de la revista oficial de Xbox. Fue hackeado y puesto a nuestro servicio. Gracias.

Descomprimimos el RAR o ZIP en una carpeta que puede llamarse "Menu" (que original que soy").

Esto es lo que veremos:



Nos metemos en la carpeta "ISO" y veremos esto:

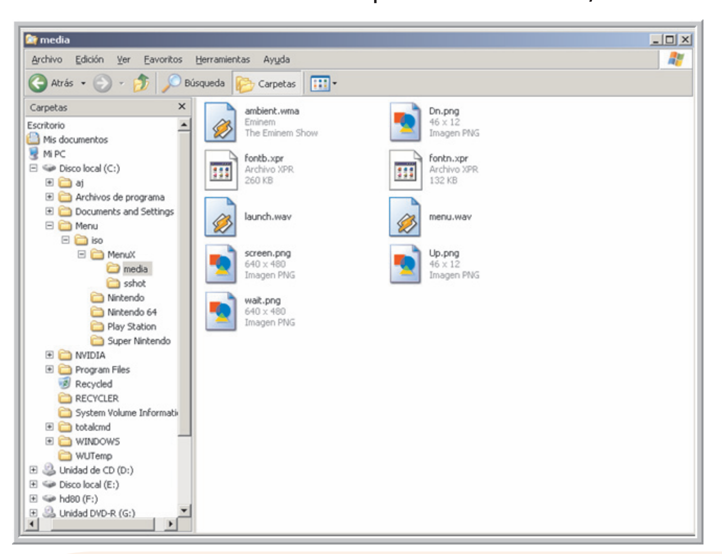


Borramos la carpeta "xbmedia", ya que esta carpeta es la que contiene el Windows Media Placer (y no nos interesa).

Aquí crearemos una carpeta por juego, solo como ejemplo yo crearé 4 carpetas que corresponden a distintos emuladores: Nintendo, SuperNintendo, Nintendo64, PlayStation.

Recordad que esto es solo un ejemplo, vosotros pondréis tantas carpetas como juegos queráis meter (siempre y cuando no superéis los 4,4 GB de tamaño total).

Ahora lo que vamos a hacer, es modificar la apariencia del menú, para ello nos vamos a la carpeta de "MenuX/Media":

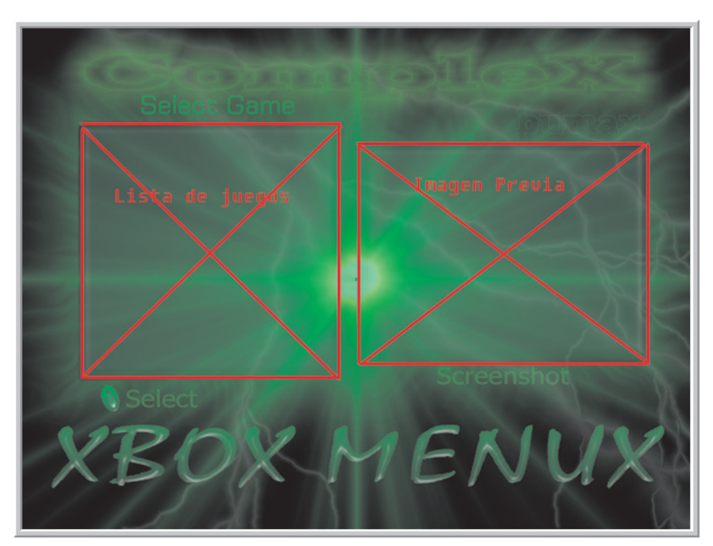


Con un editor de fotos (por ejemplo Adobe Photoshop) podemos modificar todas estas imágenes a nuestro gusto. Aquí dejo la correspondencia de cada imagen en el menú:

--> Screen.Png = El fondo del menú
--> wait.png = La imagen que sale cuando arrancas una aplicación del DVD.
--> Dn.png = Es una flecha que se coloca justo debajo del recuadro de la lista de juegos, esta apunta hacia abajo.
--> Up.png = Lo mismo que el anterior, pero dirigido hacia arriba.

Todos los archivos tienen que llamarse del mismo modo, con la misma extensión.

Screen.png es el fondo del menú consta de dos partes fundamentales, la lista de juegos y la imagen previa:



Si modificas el screen.png, ten cuidado y respeta las dimensiones de estos cuadrados que te he marcado en rojo. En todo caso, puedes hacerlos mayores que los que vienen, pero nunca más pequeños, ya que no podremos modificar el tamaño de estas celdas.

Cuando tengamos todas las imágenes listas, nos fijamos en esta misma carpeta en el archivo "ambient.wma".

Esta es la canción que sonará durante la ejecución del menú. Podemos cambiar el

archivo por otro con la misma extensión y listo, el menú ya tiene la pinta y la música que nos gusta.

Ahora nos dirigimos a la carpeta sshot, donde encontraremos la imagen previa del media player, es una imagen con extensión png y sus dimensiones son de 400 Pixeles de ancho y 300 de alto.

La borramos y ponemos aquí una imagen con extensión png (de las mismas medidas) por cada juego que vayamos a poner en el DvD. Acordaros de los nombres, ya que ahora tendremos que decirle al menú qué imagen va con cada juego.

Copiamos los juegos a sus carpetas correspondientes de la carpeta ISO.

Nos vamos a la carpeta Menú y veremos un archivo llamado config.txt, lo editamos con un editor de texto plano (por ejemplo el Bloc de Notas de Windows):

config.txt - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
TITLE=Xbox Media Player
INFO=divx/vcd/network
GFX=media.png
DIR=xbmedia
```

Como podemos ver en la imagen, tememos esto:

TITLE=Xbox Media Player
INFO=divx/vcd/network
GFX=media.png
DIR=xbmedia

Donde TITLE es el nombre del juego que saldrá en la lista de juegos.

INFO es la breve descripción del juego.

GFX es la imagen previa del juego, la/las que hemos colocado en el directorio sshot.

DIR es el directorio dentro de la carpeta ISO donde se encuentra el juego.

Entre juego y juego hay que dejar una línea en blanco.

Así quedaría el mío:

TITLE=Nintendo.
INFO=Emulador.
GFX=nintendo.png
DIR=Nintendo

TITLE=Super Nintendo
INFO=Emulador.
GFX=Snintendo.png
DIR=Super nintendo

TITLE=Nintendo 64.
INFO=Emulador
GFX=N64.png
DIR= Nintendo64

TITLE=PlayStation
INFO=Emulador
GFX=psx.png
DIR= PlayStation

Guardamos los cambios.

Arrancamos el archivo "MenuX-xdi.bat" y nos configurara el menú según le hemos dicho.

Ya esta, quemamos en un DvD el contenido de la carpeta ISO tal y como he explicado en números anteriores.



Si habéis...

Si habéis seguido los artículos anteriores publicados en la revista, podéis pasarlo por la red al HD del la consola y así cercioraros que todo ha salido bien antes de grabar el DVD.

El mes que viene veremos como crear CDs y DVDs autoinstalables.

Salu2

CURSO DE SEGURIDAD EN REDES - IDS (II)

- Vamos a avanzar en el conocimiento del SNORT
- Estudiaremos las reglas, contramedidas y MUCHO MAS

INTRODUCCIÓN

En el artículo anterior aprendimos a instalar ***snort*** e iniciamos nuestra primera incursión en el mundo de las reglas y directivas del *pre-procesador*... ahora nos toca conocerlo más a fondo.

En esta ocasión vamos a tener que realizar alguna que otra modificación en la instalación de ***snort***, de hecho nos tocará en LINUX compilar el código fuente de ***snort*** e incluir alguna característica que no se incluye en los paquetes *rpm*, no te alarmes... todo estará detallado para que consigas el objetivo.

Como recordatorio digamos que ***snort*** trabaja en varios modos, como *esniffer*, como capturador-grabador de paquetes o como *NIDS*, lo que nos ocupa es conocer las capacidades de ***snort*** en modo *IDS*, concretamente en modo *NIDS* (***NetworkIDS***)

En este (NIDS) modo podemos ejecutar ***snort*** en dos modos:

- ***Modo logging***, que grabará en disco todos los paquetes que circulan por la red o hacia o desde un *host.*

- ***Modo Alerta***, en el cual comprobará si existen reglas de comportamiento de los paquetes y en su caso nos vuelca un informe o registro de aquellos paquetes que concuerdan con las mismas, advirtiéndonos de las infracciones que se cometan.

A su vez, el modo alerta dispone de algunas opciones que se pueden lanzar desde la línea de comandos de ***snort***, estas son:

- ***Full***, es el modo por defecto y alertará de todos los paquetes que provoquen una alerta y volcará todo el contenido de los mismos
- ***Fast***, como full pero sólo registrará los mensajes de alerta, el timestamp (fecha/hora) ip origen e ip destino
- ***Syslog***, que enviará los sucesos a un servidor Syslog definido
- ***Unixsock o unsock***, que envía las alertas a un servidor "de dominio" *NIX
- ***SMB***, que envía las alertas mediante mensajes Winpopup (mensajes emergentes)
- ***None***, en el que sólo se emitirán logs, nada de alertas...

> Recordemos como se iniciaba ***snort***....
>
> ***snort –de –l directorio de logs –c directorio y fichero del archivo de configuración***
>
> Ejemplo:
> ***snort –de –l /var/snort_logs/ -c /etc/snort/snort.conf***

Si queremos utilizar algún ***modo de alertas*** como los que definimos anteriormente, la orden sería (observa la **opción –A** que es quien indica el modo de alerta):

snort –de –A fast –l /var/snort_logs/ -c /etc/snort/snort.conf

Hay algunas matizaciones con los modos *Syslog y SMB*, más adelante las descubriremos, por ahora sólo es necesario que recuerdes que se pueden usar y que existen.

Si queremos usar ***snort*** sin generar alertas bastaría utilizar la sintaxis:
snort –de –l /var/snort_logs/

Para complicar más la cosa, podemos "*formatear*" la salida de logs… es decir, darle un aspecto para su posterior lectura… entre los modos de presentación más habituales están:

CSV (Comma Separated Values), es un modo habitual para observar los registros desde una hoja de cálculo o base de Datos

Syslog, para personalizar la salida hacia el servidor Syslog

Database, para escribir la salida a Bases de Datos tipo MySQL, ODBC, ORACLE, etc..

Null, no crea archivos de logs pero sí alertas

Tcpdump o Windump, formato de salida para aplicaciones tcpdumo o windump

SnmpTrap, enviar registros SNMP

Unified, Formato de salida binario que puede ser leído por diversas aplicaciones, es el método de salida más rápido

XML, pues lo imaginas.... formatos de salida en lenguaje XML

Bueno, esto nos servirá como introducción para más adelante, cuando nos toque abordar los *plug-ins de salida*... al igual que antes... con que recuerdes que no todo es texto y con el mismo formato es suficiente.

Tras la introducción... pasemos a lo que nos ocupa en este artículo: ***Las Reglas***

LAS REGLAS de las REGLAS....

Ya nos hicimos a la idea de para qué puede servir el definir reglas en el ***IDS***, las reglas son el "*alma*" en la detección de intrusos... aquellos paquetes que cumplan (o no cumplan) las reglas pasarán o no pasarán... mejor dicho... se registrarán o no se registrarán.

Las reglas que ***snort*** puede aplicar al tráfico de red se basan en las "***cabeceras***" de los paquetes y en el "***cuerpo***" o contenido de los mismos y en el ***control de flujo***, por tanto podremos aplicar **reglas de cabecera**, **reglas de contenido** y **reglas de control**.

LAS VARIABLES:

Antes de meternos de lleno en las reglas, vamos a ver unas cuantas cosas. Para que ***snort*** pueda comparar esas reglas con "*algo*" o desde "*alguna parte*" se utilizan variables, la sintaxis genérica de las variables es:

var nombre_de_variable valor_de_variable

Todas las variables se definen en el archivo de configuración (***snort.conf***) y se procesan "*de arriba abajo*", y como es comprensible, deben estar definidas e inicializadas antes de usarse.

Ejemplos:

var DNS_SERVER 172.28.0.98
var INTERNAL_NET 172.28.0.0/16
var INTERNAL_NETS [172.28.0.0/16, 192.168.1.0/24, 10.0.0.0/8]

Si queremos asignar el contenido de "*otra*" variable a una nueva, usaremos:

var nueva_variable $variable_anterior

Esto es:

var MI_DNS $DNS_SERVER
var ESTA_ES_MIRED $INTERNAL_NET

Estarás pensando qué pasaría en estos últimos casos si las variables que actúan como valores de las nuevas no estuvieran previamente definidas... un error... para ello ***snort*** puede utilizar el siguiente método:

var ESTA_ES_MIRED $(RED_INTERNA:172.28.0.0/16)

Esto significa que si ***$RED_INTERNA*** no está definida previamente, se asignará el valor 172.28.0.0/16 a la variable ***ESTA_ES_MIRED***

Hasta se pueden lanzar mensajes de error para que "*avise*" de los mismos... por ejemplo:

var ESTA_ES_MIRED $(RED_INTERNA:? Vic_Thor, define RED_INTERNA)

Echa un vistazo al archivo ***snort.conf*** para ver las variables que usa... entre ellas encontrarás estas con estos valores (o parecidos):

Variable name	IP Address/Range
HOME_NET	any
EXTERNAL_NET	any
DNS_SERVERS	$HOME_NET
SMTP_SERVERS	$HOME_NET
HTTP_SERVERS	$HOME_NET
SQL_SERVERS	$HOME_NET
TELNET_SERVERS	$HOME_NET
SNMP_SERVERS	$HOME_NET
HTTP_PORTS	80
SHELLCODE_PORTS	!80
ORACLE_PORTS	1521
AIM_SERVERS	[64.12.24.0/24,64.12.25.0/24,64.12.26.14/...
RULE_PATH	../rules

Algunos valores especiales para la asignación de variables son:

any Cualquier valor

! Negación, por ejemplo **!80** es NO el 80,

LAS REGLAS

Tras definir las variables tendremos que decidir qué reglas vamos a incluir en el *IDS* y las directivas del *pre-procesador* que queramos...

Normalmente las reglas a ser utilizadas se declaran mediante las líneas ***include*** y las directivas del *pre-procesador* mediante las líneas ***preporcessor***:
Ejemplo sacado del archivo de configuración original

```
(snort.conf)RULE_PATH /etc/snort/rules
.....
preprocessor stream4_reassemble
preprocessor stream4: disable_evasion_alerts
......
include $RULE_PATH/local.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
......
```

Lógicamente el hecho de incluir una regla en el archivo de configuración no lo es todo... lo importante es conocer qué tiene esa regla. En el archivo de configuración simplemente se le dice a ***snort*** que deseamos utilizar tal o cual regla y será en los archivos especificados donde tendremos que definir el comportamiento de la regla.

Lo mismo podríamos aplicar al *pre-procesador*... en la siguiente sección nos ocuparemos de él, ahora "*toca*" reglas....

Ya sabemos que ***snort*** valida las reglas de varias formas, consulta las cabeceras de un paquete de datos, consulta el contenido de ese paquete, controla el flujo de datos... o todas a la vez...

volvemos con las analogías... las cabeceras serían el equivalente de las señas, remitente, etc. Del sobre de una carta o correo postal, mientras que las reglas de contenido "*abrirían el sobre*" y buscarían dentro para "*leer*" lo que dice y actuar en consecuencia...

Las reglas de Cabecera inspeccionan:

- Protocolos usados
- Puertos origen y/o destino
- Direcciones IP origen y/o destino

Las reglas de Contenido inspeccionan:

- La totalidad o parte de la información que no es cabecera.

Las reglas de Control de flujo inspeccionan:

- La dirección del tráfico (del cliente al servidor, del servidor al cliente)
- El número de paquetes transmitidos
- El tiempo de cada sesión
- Las contramedidas o reacciones
- ....

Las reglas se definen de una forma muy sencilla:

Acción Protocolo IP/red origen Puerto origen Dirección IP/red Destino Puerto destino Contenido control de flujo

Vamos a detallarlo:

Acción puede ser: pass, log, alert, dynamic o activate

Protocolo puede ser: ICMP, TCP, IP o UDP, hay otros pero fundamentalmente estos.

Origen y/o Destino son: Direcciones IP individuales o direcciones de red en notación CIDR

Dirección es decir si va hacia el origen o hacia el destino

Contenido es todo aquello que "contiene" el paquete, es decir los datos de que trasportan los protocolos.

Control de Flujo: Tiempo transcurrido, número de paquetes, corte de la conexión...

Vamos a ver un ejemplo de una regla "tipo":_

Acción: alert
Protocolo: tcp
IP/red origen: 172.28.0.0/16
Puerto Origen: cualquiera
Dirección: Desde el origen al destino
IP/red destino: Cualquiera
Puerto destino: 80

Lo que nos quedaría así:

alert tcp 172.28.0.0/16 any -> any 80

Esta regla generaría una alerta al usar el protocolo TCP cuyo origen del paquete sea desde la red 172.28.0.0/16 y desde cualquier puerto hacia cualquier red cuyo destino sea el puerto 80.

En otras palabras, si cualquier PC de la red 172.28.0.0/16 navega hacia cualquier servidor web que utilice el puerto 80, será logueado y generará una alerta del mismo.

Como estarás pensando, esa regla no es que sea precisamente efectiva, puesto que tendrá un trabajo enorme debido a que todas las sesiones de navegación de la red se volcarán en los logs y alertas.

Vamos a afinar un poquito más... pongamos que queremos alertar de sólo aquellas "*navegaciones*" hacia páginas de sexo....

Ahora todas las páginas visitadas en cuya dirección web aparezca la palabra sexo serán las registradas, las otras no...

alert tcp 172.28.0.0/16 any -> any 80 (content: "sexo";)

Claro, que si el servidor pornográfico no utiliza el puerto 80 tampoco saltaría la alerta, la solución será fácil, ¿no?

alert tcp 172.28.0.0/16 any -> any any (content: "sexo";)

Ahora alertará de todos los accesos por TCP en los que intervenga la palabra sexo, independientemente del Puerto destino (*any*)

Bueno, es sólo un ejemplo para ir conociendo algo más de las reglas, realmente son bastante más complicadas y potentes, veamos TODO acerca de las reglas... y empecemos por las más sencillas... las reglas de Cabeceras.

Reglas de Cabecera

Acción	Descripción
pass	Ignora el paquete... simplemente lo deja pasar
log	Registra el paquete y lo graba en el archivo de logs que se indicó
alert	Genera una alerta y la guarda en el archivo alerts del directorio indicado
dynamic	Utilizado junto con activate, tiene varios usos, entre ellos la capacidad de logear un número de paquetes determinado o durante un tiempo, etc...
activate	Activa una regla dynamic, ahora no es el momento de entenderlo todo bien, luego con ejemplos lo verás más claro.

Protocolos

Sería ímprobo detallar cada uno de ellos y realizar una descripción pormenorizada uno a uno, sirvan unas breves líneas para expresar sus cometidos. Los protocolos que se pueden usar son:

Protocolo	Breve Descripción
TCP	Protocolo de transporte, confiable, orientado a conexión extremo a extremo, control de errores, utiliza puertos para seguir las conexiones
UDP	Protocolo de transporte no confiable, no orientado a conexión, sin control de flujo y sin corrección de errores, utiliza puertos para seguir las conexiones
IP	Protocolo de red que permite encaminar y llegar hasta la red origen/destino
ICMP	Protocolo cuya función es la de notificar eventos, proporciona un medio de transporte para que los equipos se envíen mensajes de control y error. ICMP no está orientado a la corrección de errores, sólo a su notificación y solamente informa de incidencias en la entrega de paquetes o de errores en la red en general, pero no toma decisión alguna al respecto. Esto es tarea de las capas y protocolos superiores.
802.11	Protocolo utilizado para redes sin cable. Wireless
HTTP	Protocolo para aplicaciones cliente servidor en entornos web
ARP	Protocolo encargado de establecer las relaciones entre las direcciones MAC e IP

Direcciones IP y direcciones de Red.

Se usan para definir la IP origen y/o IP destino... también puede ser una red o subred completa.

Las direcciones de red en ***snort***, se indican en notación ***CIDR*** (*Classless InterDomain Routing, Enrutamiento sin clase*) esto es, que en lugar de usar las máscaras de subred en la notación decimal punteada (255.255.255.255) se usa el formato /xx donde xx es un número que equivale a la mascara de subred.

Le pisaremos "un poquito" el terreno a nuestro compañero que está haciendo el curso de TCP/IP y brevemente repasemos cómo es una dirección IP :)

Componentes de una dirección IP

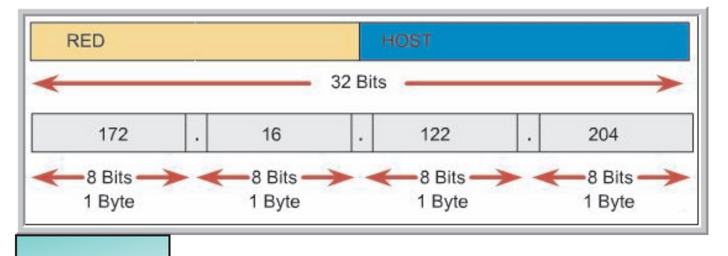


Figura 1. Componentes de una dirección IP

- En toda dirección IP existe una parte de Red y una parte destinada para el *host*
- Las direcciones IP (Ipv4) son de 32 bits, representadas en 4 grupos de 8 bits cada una

El número de red de una dirección IP identifica la red a la cual se encuentra adherido un dispositivo.

La parte de *host* de una dirección IP identifica el dispositivo específico de esta red.

Como las direcciones IP están formadas por cuatro octetos separados por puntos, se pueden utilizar uno, dos o tres de estos octetos para identificar el número de red.

De modo similar, se pueden utilizar hasta tres de estos octetos para identificar la parte de *host* de una dirección IP.

Debido a esta característica, las direcciones IP se suelen separar en clases:

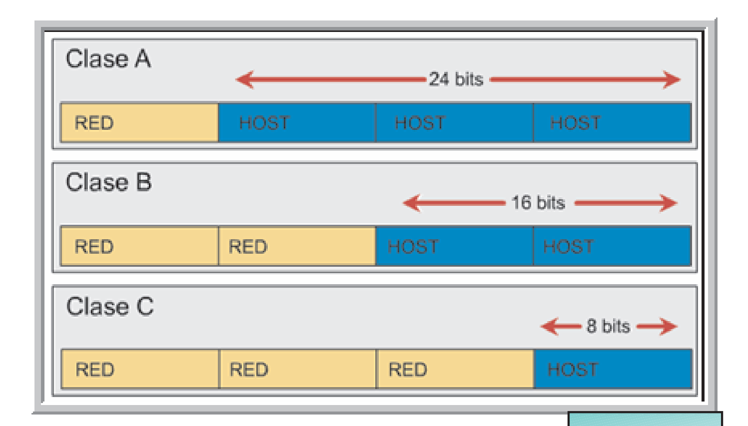


Figura 2. Componentes de una dirección IP

Como muestra la figura anterior, disponemos de tres Clases de Direcciones IP A-B-C, bueno también existen D y E, pero de momento no para nosotros.

CLASE A

Una manera fácil de reconocer si un dispositivo forma parte de una red Clase A es verificar el primer octeto de su dirección IP, cuyo valor debe estar entre 0 y 127. Con una excepción, todas aquellas direcciones que comiencen por 127.xxx.xxx.xxx, que son consideradas como direcciones de ***loopback*** (el tema **loopback** ya se explicó en anteriores números de la revista).

CLASE B

Las direcciones IP Clase B siempre tienen valores que van del 128 al 191 en su primer octeto.

CLASE C

Las direcciones IP Clase C siempre tienen valores que van del 192 al 223 en su primer octeto.

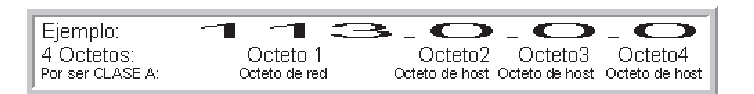
¿Qué es una Dirección de Red?

Definición: **Una Dirección de Red es una dirección IP especial que contiene ceros binarios en todos los bits del host.**

¿Cómo? ¿Qué has dicho? ¿Ein?

Tranquilo, se verá mejor con un ejemplo: En una **red de Clase A**, la **IP 113.0.0.0** (por ejemplo) está reservada, ningún host tendrá esa IP, tu PC jamás podrá tener esa IP, NADIE puede tener esa IP. Esa IP es la llamada **dirección IP de la red**.

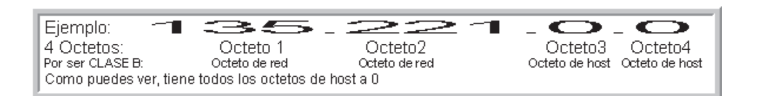
Ejemplo de Dirección de red:



Fíjate en lo que hemos dicho antes y fíjate en el ejemplo. Hemos hablado de una **IP** que contiene ceros binarios en todos los bits de **host**... Fíjate que en una **red de Tipo A** los **host** corresponden a los tres octetos finales (24 bits) y la **IP** que hemos puesto de ejemplo cumple esa propiedad puesto que sus tres octetos finales son ceros :)

113.0.0.0 es la dirección IP de la red que contiene, por ejemplo, el host 113.1.2.3 (y el 113.67.24.5 y el 113.4.5.6, etc). Otro ejemplo de Dirección de Red de Clase A sería la IP 87.0.0.0, que contendría el host 87.24.56.12 (y el 87.65.65.51, etc).

Venga, ponme un ejemplo de Dirección de Red de Clase B... venga, no es muy difícil: 135.221.0.0



La dirección IP de la red se encuentra reservada. Nunca se usará como dirección para un dispositivo (ordenador, router, etc) conectado a ella.

¿Qué es una dirección Broadcast?

Si un ***host*** (por ejemplo tu PC) desea comunicarse con todos los dispositivos de una ***red***, sería prácticamente imposible escribir la dirección IP para cada dispositivo (que trabajo más desesperante, hay muuuuuuuuuchas direcciones posibles).

Para esa tarea existe un método abreviado: **Si queremos enviar datos a todos los host de la red, utilizaremos una *dirección* de *broadcast*.** Un **broadcast** se produce cuando una fuente envía datos a todos los dispositivos de una red, es muy similar al envío de correo masivo.

Bueno, venga... ponme un ejemplo de dirección broadcast.

El curso de TCP/IP va por otro lado, pero bueno... te lo explico "plan rápido". Es lo mismo que en el tema de las direcciones de red pero con todos los bits puestos a 1, es decir, son direcciones tipo 113.255.255.255 o 135.221.255.255 (antes en lugar de 255 teníamos ceros, y el 255 sale de poner todos los bits a 1... ya tocamos las conversiones binarias en otros números de la revista).

Y por si no...

Y por si no te he cansado lo suficiente, para los más curiosos (para que investigues en el google, www.google.com) te diremos que los *hosts* en una red sólo pueden comunicarse directamente con dispositivos que tienen el mismo ID de red y que los enrutadores de IP no propagan ("reenvían") los paquetes de difusión de red (broadcast).

Se está preparando un **número especial para AGOSTO** donde se explicará MUY DETALLADAMENTE todo esto.

Redes, subredes y Máscaras de red

Las redes IP son arquitecturas lógicas, todas las máquinas que forman parte de una red tienen algo en común, su mismo Identificador de Red.

Las IP de Red las asigna el administrador, pueden cambiar y de hecho cambian con frecuencia, basta con trasladar un equipo de un sitio a otro para que pueda ser preciso cambiar la dirección de red e incluso la dirección de *host*.

Todo lo contrario ocurre con las **direcciones MAC, estas son "*permanentes*"** y no suelen cambiar a menos que cambie el Hardware del *host*.

Veamos una tabla que ilustra el número de redes que puede existir para cada clase.

Clase	Dirección de Broadcast	Dirección de Red	Host disponibles	Número de Redes
A	xxx.255.255.255	xxx.0.0.0	16.777.214	254
B	xxx.xxx.255.255	xxx.xxx.0.0	65.534	65534
C	xxx.xxx.xxx.255	xxx.xxx.xxx.0	254	16.777.214

Deberías leer este hilo del foro, en él ***moebius,*** explica magníficamente y con numerosos ejemplos *¿Qué es eso de las máscaras de subred*?, No lo olvides, precisamente por que existe ese link yo no voy a extenderme mucho en el subnetting, además, eso me haría "pisar" el magnífico curso de TCP/IP que se está impartiendo en esta revista ;)

http://www.hackxcrack.com/phpBB2/viewtopic.php?t=8597

Resumiendo, que para nuestro ***snort***... el formato para explicarle cuantos pc's forman una red o subred es así:

Formato CIDR	Formato decimal punteado
/8	255.0.0.0
/9	255.128.0.0 * puede ser inválida
/10	255.192.0.0
/11	255.224.0.0
/12	255.240.0.0
/13	255.248.0.0
/14	255.252.0.0
/15	255.254.0.0.
/16	255.255.0.0
/17	255.255.128.0
/18	255.255.192.0
/19	255.255.224.0
/20	255.255.240.0
/21	255.255.248.0
/22	255.255.252.0
/23	255.255.254.0
/24	255.255.255.0
/25	255.255.255.128
/26	255.255.255.192
/27	255.255.255.224
/28	255.255.255.240
/29	255.255.255.248
/30	255.255.255.252
/31	255.255.255.254
/32	255.255.255.255

Puertos Origen y Puertos Destino

Los puertos es otra de las cosas que deberías tener claro, para lo que sirven y lo que son.

Un puerto TCP ó UDP **es la ubicación o medio que se abre para enviar/recibir mensajes usando los *servicios TCP o UDP*, ese puerto es un número entre 0 y 65535.**

La IANA **(*Internet Assigned Numbers Authority*) asigna números de puertos conocidos a protocolos** *TCP ó UDP* **que pueden ser usados y reserva del 0 al 1023 para este tipo de servicios.**

Un ejemplo es el "archiconocido" puerto 80, que se utiliza para navegar por Internet de Web en Web. Cuando en el Internet Explorer pones www.google.com se asume que lo que has querido decir es www.google.com:80 (80 es el puerto, prueba a ponerlo en tu navegador www.google.com:80 y verás que funciona perfectamente; pero no pongas otro puerto o no funcionará :))

A grandes rasgos, los números de puerto tienen los siguientes intervalos asignados:

- Los números inferiores a 255 se usan para aplicaciones públicas.
- Los números del 255 al 1023 son asignados a empresas para aplicaciones comerciales.
- Los números superiores a 1023 no están regulados.

Convencionalmente usamos determinados números de puerto asociados a ciertas aplicaciones o servicios, pero nada nos impide alterar ese "*convenio*" siempre y cuando cliente y servidor se pongan de acuerdo y se dirijan las peticiones por los puertos "*acordados*.

Entre los servicios y puertos más comunes, están:

Puerto	Servicio o Aplicación
20 y 21	FTP
25	SMTP
53	DNS
80	WEB
110	POP
137, 138, 139	NetBIOS
135	RPC
445	SMB
3306	MySQL

Bueno y un sin fin de ellos más, seguro que por Internet o por nuestros foros encontrarás más información de cada uno de ellos, lo que son y para qué se usan... además, hasta los hay propios de UDP, de TCP o para ambos... vamos que tampoco hay que saberse la lista de los 65000 y pico como si fuesen la lista de los *Reyes Godos*....

Por otra parte y volviendo a las reglas de ***snort***, es posible explicar a nuestras reglas si queremos utilizar más de un puerto, imagina que quieres registrar el tráfico de nuestra red hacia otras redes pero sólo de aquellas comunicaciones que usen puertos "*por encima*" del 1024, sería así:

log tcp 172.28.0.0/16 any -> any 1024:65535

o también:

log tcp 172.28.0.0/16 any -> any 1024:

es decir, el carácter dos puntos (**:**) podemos usarlo como un separador de rangos, de forma que si aparece antes del número lo interpretaremos como "*hasta*" y si aparece después del número de puerto lo interpretaremos como "*desde*", ejemplo:

1025: desde el 1025
:1024 Hasta el 1024
1:1000 desde el uno al 1000

La palabra ***any*** significa "*cualquiera*" y recuerda que podemos usar la exclamación (**!**) para negar cualquier cosa, en este caso un puerto, por ejemplo

log tcp 172.28.0.0/16 any -> any !80

Este caso deberíamos leerlo como cualquier Puerto que **no sea** el 80.

Dirección o Flujo de datos

En las alertas de ***snort*** se define la Dirección de tres formas:

-> Hacia el Destino
<- Desde el destino
<> Bidireccional, entre el origen y el destino

Es fácilmente comprensible, pongamos esta regla:

alert tcp 172.28.0.0/16 any -> any 80

Alertará de los paquetes tcp que fluyan **desde** la red 172.28.0.0/16 **hacia** cualquier otra y con destino el puerto 80.

Observa que si un equipo de otra red "*entrase*" en la nuestra aunque sea por el puerto 80, esos paquetes no se registrarían, para ello tendríamos que haber usado <-

Pero claro, si usásemos esa notación, no se registrarían los que nuestra red emite... o sea, que si queremos todos, desde fuera o desde dentro, usaríamos <>

Antes de pasar a las reglas de contenido, veamos otra forma útil de definir las reglas, el uso de los **métodos Activate y Dynamic**

Empecemos con un ejemplo para "*verlo*" más claro....

```
activate tcp any any -> any 80 (activates: 80; msg: "OJO, Petición Web";)
dynamic tcp any any -> any 80 (activated_by: 80; count: 25;)
```

Bien, lo primero es que "*corté*" la línea antes de ***dynamic***... esto no es así, sólo para mejorar "*la visualización*" de la regla, realmente iría todo "*seguidito*"

Lo que sí es importante es que exista correlación entre el número que sigue a ***activates:*** y el número que sigue a ***activated_by***

Utilicé el número 80 pero ***no lo confundas con el puerto del protocolo***, no tiene nada que ver, simplemente la regla ***dynamic*** se lanza gracias a lo que diga ***activate*** y para establecer la relación entre ellas, ambas deberán utilizar un número de regla de activación.

La función de esta regla es:

Se genera una alerta y registro con el mensaje "*OJO, Petición Web*" cuando cualquier equipo de la red realiza una petición TCP hacia cualquier otra red por el puerto 80, hasta aquí es igual que siempre, la diferencia está en que al incluir ***dynamic***, además de eso, le pedimos a ***snort*** que "*continúe*" registrando los siguientes 25 paquetes de esa misma conexión para "*observar*" lo que hace el cliente que activó la alerta.

Es bastante potente... podemos "*vigilar*" y seguir "*los pasos*" de quien rompe las reglas que establecimos... de ese modo podemos monitorizar en tiempo real que demonios está haciendo nuestro "*sospechoso*"

Si combinamos "*sabiamente*" todo lo que ya llevamos aprendido podemos empezar a monitorizar el tráfico de nuestra red "*a medida*" incluyendo sólo aquellas reglas que se necesiten.

En el ejemplo anterior utilizamos "*algo nuevo*" hasta ahora, ***msg: "Texto"***, no le prestes mayor importancia, simplemente será un texto descriptivo que se guardará junto con las alertas y registros, esa forma es parte de las reglas de contenido que veremos a continuación, pero antes vamos a realizar un par de prácticas para afianzar lo aprendido...

Práctica 1. Detectar puertos Extraños y conexiones sospechosas.

Imaginemos que somos los administradores de una red y queremos saber varias cosas:

1º) Qué IP's internas intentan acceder a servicios externos que no son aplicaciones públicas, es decir, aquellas máquinas que intentan abrir conexiones con equipos remotos que utilizan puertos superiores al 1024, emule, KaZa, e-donkey....

2º) Qué ip's internas acceden a páginas webs de contenido erótico o pornográfico, bueno y ya puestos páginas de crackers, hackers y "*esos mundos*"

3º) Pongamos que nuestra política de red es la de no permitir a los usuarios que utilicen nuestros accesos para sus correos privados, es decir, queremos averiguar quienes acceden a servidores de correo tipo *hotmail, yahoo, wanadoo*, etc...

Bueno, podríamos seguir, con estos ejemplos imagino que ya sabes por donde voy....

El seguimiento de esta práctica la haremos con un ***snort*** corriendo en un equipo *LiNUX*, por última vez te repito la diferencia en la estructura de directorios entre ambos sistemas operativos en cuanto a ***snort*** se refiere:

Directorio / paquete	Windows	LiNUX
Ejecutables del programa	C:\snort\bin\snort.exe	/sbin/snort
Archivo de configuración	C:\snort\etc\snort.conf	/etc/snort/snort.conf
Directorio de Reglas	C:\snort\rules	/etc/snort/rules/
Alertas y logs	A definir, por defecto C:\snort\logs	A definir, por defecto /var/log/snort/

Advierto que los directorios y/o rutas mostradas en la tabla anterior pueden cambiar dependiendo de cómo lo decidimos en el momento de la instalación... asumiré que son estos u otros nuevos que nos crearemos para el caso.... Así que al tajo...

Primero definamos nuestro escenario y recursos....

a.- Disponemos de una LAN con dirección de red ***172.28.0.0/16*** y varios equipos en ella

b.- ***El IDS*** corre en una máquina LiNUX de dirección IP ***172.28.0.200/16***

c.- Crearemos un nuevo **archivo de configuración llamado mi_snort.conf** y lo guardaremos dentro de los directorios mostrados en la tabla anterior.

d.- Crearemos 1 archivo con las ***reglas particulares***, lo llamaremos ***mis_reglas*** y lo guardaremos en el directorio de reglas como indica la tabla

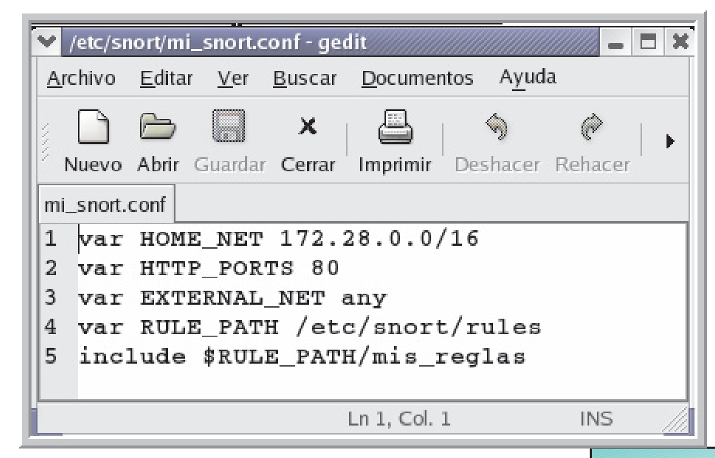
e.- **Las alertas, sucesos, logs, etc...** los guardaremos en el directorio ***/var/mis_logs***, para ello habrá que crearlo previamente.

Accedemos al directorio ***snort*** para crear el nuevo archivo de configuración:

```
root@linux-rh:/etc/snort
Archivo  Editar  Ver  Terminal  Ir  Ayuda
[root@linux-rh root]# cd /etc/snort
[root@linux-rh snort]# gedit mi_snort.conf
[root@linux-rh snort]#
```

Pantalla 1. Crear un nuevo archivo de configuración llamado mi_snort.conf

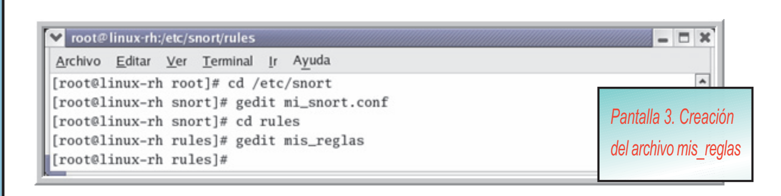
Una vez abierto el editor ***gedit***, escribimos estas líneas para crear el nuevo archivo de configuración:



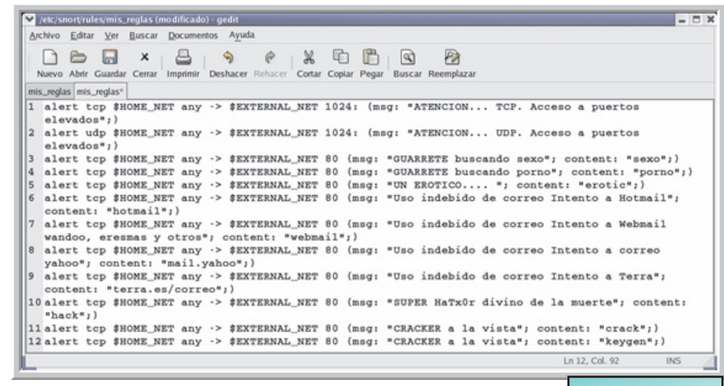
Pantalla 2. Contenido del archivo de configuración mi_snort.conf

Y lo guardamos

Ahora accedemos al directorio donde están las reglas y crearemos del mismo modo que antes el archivo *mis_reglas*



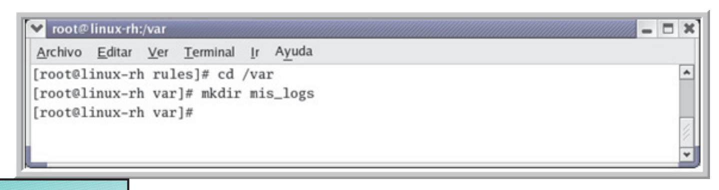
Pantalla 3. Creación del archivo mis_reglas



Pantalla 4. Contenido del archivo mis_reglas

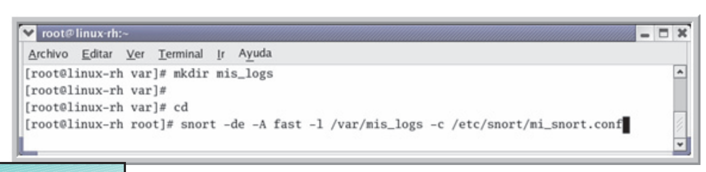
No creo que haga falta comentar mucho más estas reglas, con todo lo dicho hasta ahora y estos ejemplos "*repetitivos*" no deberías tener ningún problema en ello...

Ahora ***cambiamos al directorio /var y creamos el archivo mis_logs*** tal y como se decía en el último paso a realizar... no es la única forma, hay otras pero ya sabes... poco a poco que no hemos hecho más que empezar.



Pantalla 5. Creación del directorio donde se guardarán los logs. /var/mis_logs

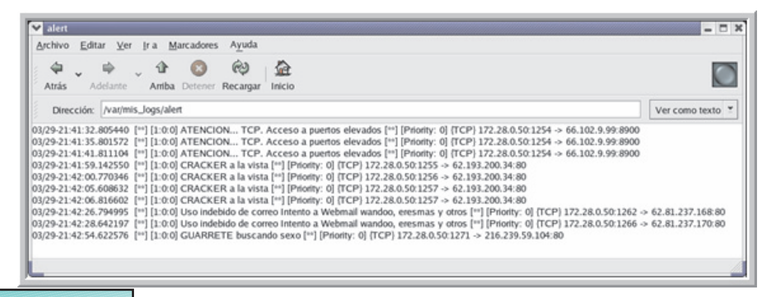
Bien, ya lo tenemos todo, ahora sólo hace falta **iniciar *snort*** y darle los parámetros adecuados para que busque nuestro archivo de configuración y cargue las nuevas reglas....



Pantalla 6. Ejecucion de Snort desde la linea de comandos

Vamos a coger cualquier equipo de la red y **violemos las reglas**, todos los ejemplos que vienen a continuación los infringió el mismo equipo, el **172.28.0.50**, que accedió por un **puerto superior al 1024**, **buscó una página de sexo y accedió** a su correo personal a través de webmail de wanadoo y luego se pasó por los **foros de HackXCrack**

Este sería el contenido del archivo ***alert en /var/mis_logs***



Pantalla 7. Contenido del archivo alert en /var/mis_logs

Recuerda que dentro del directorio ***/var/mis_logs*** se crearán también otras carpetas o directorios con los contenidos de los paquetes "*prohibidos*" tal y como vimos en el artículo del mes pasado...

Y otra cosa... si te estás preguntando si el propio ***snort*** es capaz de "*abortar*" esas acciones, la respuesta es **SI**... *espera a ver cómo avanzamos y lo conseguirás implementar... registrará los paquetes y si lo deseas "corta" la comunicación y más....*

Reglas de contenido

Acabamos de mostrar un ejemplo en el uso de las reglas de *snort* con una "*introducción a las reglas de contenido*", veamos ahora más posibilidades de éstas:

Contenido de los paquetes (content:)

Puede comprobar contenidos en formatos ***Ascii (texto)***, ***binarios-hexadecimales o ambos***.

Uso: *content: "contenido a examinar dentro del paquete"*

Ejemplos:

alert tcp any any -> any any (content: "sexo";)
alert tcp any any -> any any (content: "|0101 EFFF|";)
alert tcp any any -> any any (content: "|0101| /etc/password |EFFF|";)

Explicaciones:

Generan una alerta si el contenido coincide con cualquiera de las expresiones que siguen a la "*instrucción*" ***content:***

Muchos ataques conocidos, virus, exploits, rootkits, etc... son bien conocidos, incluyen lo que se llama un "*payload*" o *carga útil del paquete*... con ***snort*** podemos definir esos *payloads* en las reglas ***content:*** para que generen alertas o *logs* cuando esto ocurre.

Profundidad (depth:)

Establece el número de bytes que serán analizados dentro del contenido del paquete.

Uso: *depth: número de bytes*

Ejemplos:

alert tcp any any -> any any (content: "sexo"; depth:100;)

Explicaciones

Con profundidad queremos definir la cantidad de bytes que examinará ***snort*** en una regla en busca del *payload* y/o contenido de los paquetes, en el ejemplo expuesto examinará los primeros 100 bytes del paquete buscando la palabra sexo....

Es una opción útil puesto que ***snort*** no analizará en todo el contenido del paquete, sólo en los bytes indicados, con lo que evitamos un trabajo en exceso.

Desplazamiento (offset:)

Establece el número del byte a partir del cual empezará a ser serán analizado el contenido del paquete.

Uso: *offset: número de bytes*

Ejemplo:

alert tcp any any -> any any (content: "sexo"; depth:100; offset: 8)

Explicaciones

Comenzará el análisis a partir del byte número 8 en lugar desde el principio (byte cero) que sería la opción por defecto... No es una opción muy usada.

Ignorar mayúsculas/minúsculas. (nocase;)

Si no se utiliza la opción ***nocase***, ***snort*** analizará exactamente el contenido de la opción ***content***:, lo que se llama "*case sensitive*", si deseamos ignorar la diferencia entre mayúsculas y minúsculas usaremos ***nocase***.

Uso: *nocase;*

Ejemplos:

alert tcp any any -> any any (content: "sexo"; nocase;)

Explicaciones

Alertará de contenidos en los que sexo, Sexo, SEXO, sExO, etc.. estén presentes, si bien, a partir de la versión 2.1 de snort, ya se toma como opción por defecto.

Contenido URI: uricontent:

Anteriormente vimos que con la opción content, se examinaba el contenido de cualquier paquete, con uricontent se examinará únicamente el contenido de la dirección URL de una visita a un servidor web, con ello se "*ahorra*" procesar todo el contenido del paquete y buscará solamente en la dirección web escrita

Uso: *uriconent: "contenido uri"*

Ejemplos:

alert tcp any any -> any 80 (uricontent "%255c";)

Explicaciones

Analiza si el contenido de una sesión web, se escribió la secuencia %255c como parte de la dirección web.

Mensaje de alerta(msg:)

Escribe un mensaje dentro del archivo ***alert*** que generará la alerta

Uso: *msg: "Texto a mostrar"*

Ejemplos:

alert tcp any any -> any 23 (content: "login failed"; session printable;nocase; msg:"ATENCION Intento de acceso por TELNET";)

Explicaciones

Escribe *ATENCIÓN intento de acceso por TELENET* como cabecera de en el archivo de alertas generado.

Aunque existen otras opciones en el contenido de las reglas, éstas que hemos comentado son las más usadas, puedes revisar el resto en:

http://www.snort.org/docs/snort_manual/node14.html

No todas las opciones que se pueden aplicar a las reglas son de cabecera de paquetes o de contenido de los mismos, hay más... hay unas cuantas que son muy interesantes (otras menos) que se pueden utilizar tras la detección (*Post-detection)* o para controlar el flujo de los paquetes (***flow***)

REGLAS DE CONTROL

Sesión (session:)

Permite registrar en "*texto claro*" los datos que transmite el protocolo y los vuelca por pantalla con el formato de salida elegido.

Uso: *session: printable ó session: all*

Ejemplos:

alert tcp any any -> any 23 (content: "login failed"; session printable;nocase;)

Explicaciones

Parecido a *content* pero para protocolos y puertos especificados, el ejemplo anterior generará una alerta y un archivo llamado ***SESSION:puerto_origen-23*** que guardará el resultado de una sesión *telnet* fallida...

Control del flujo (flow)

Establece la dirección de los paquetes, por ejemplo si van desde el cliente al servidor, desde el servidor al cliente, hacia el servidor, etc...

Uso: *flow: opción de control*

Disponemos de varios controles en el sentido del tráfico:

Opción de control	Descripción
to_server	Se activa si el paquete se envía al servidor
from_server	Se activa si el paquete viene desde el servidor
to_client	Se activa si el paquete se envía al cliente
from_client	Se activa si el paquete viene del cliente
only_stream	Se activa sólo si el paquete se reconstruye tras una sesión establecida
no_stream	Se activa aunque la sesión no se haya establecido
established	Se activa cuando el protocolo TCP establece una sesión (ACK)
stateless	Se activa independientemente del estado de la sesión (FIN, RST, PSH...)

Ejemplos:

alert tcp any any -> any 23 (content: "login failed"; session printable;nocase; flow:from_server;)

Explicaciones

Se activará la alerta cuando el paquete provenga del servidor telnet

Es una opción muy utilizada, sobre todo para registrar diferentes ataques del tipo D.o.S. en los que las sesiones no terminan de establecerse por completo.

Registrar (logto:)

logto le dice a ***Snort*** que registre todos los paquetes que accionen esta regla y que vuelque su contenido a un fichero de registro de salida. Esto es especialmente práctico para combinar datos de actividades en ***escaneo de puertos***, ***exploraciones de cgi, etc***. ***Esta opción no trabaja cuando Snort está en modo de registro binario.***

Uso: *logto: "Ruta/archivo.extensión"*

Ejemplos:

alert tcp any any -> any 23 (content: "login failed"; session printable;nocase; logto:"/var/mis_logs/autenticacion_telnet";)

Explicaciones

Guardará un registro en el ***archivo autenticacion_telnet*** dentro del directorio ***/var/mis_logs***, es alternativo al ***parámetro –l*** en la inicialización de ***snort***, gracias a esta opción podemos "*separar*" unas alertas de otras o seleccionar archivos específicos por su especial relevancia.

Etiquetas (tag:)

Tag permite registrar un número de paquetes adicionales si una alerta se dispara.

Uso: *tag: host/session , cantidad, paquetes/segundos, origen/destino*

Ejemplos:

alert tcp any any -> any 23 (tag:host,10, packets;)
alert tcp any any -> any 23 (tag:host,20, seconds;)

Explicaciones

Registrará 10 paquetes (primera sintaxis) o durante 20 segundos (segunda sintaxis) el tráfico *telnet* que se establezca entre servidor y cliente.

CONTRAMEDIDAS Y ACCIONES

El grupo de opciones que se detallan a continuación permitirán tomar a ***snort*** una determinación acerca de lo que debe hacer con la conexión que disparó la alarma.

Estas opciones se encuadran dentro de las llamadas "***Flex Resp***" *respuestas flexibles* y para que puedan ser utilizadas debemos realizar previamente algunas modificaciones en el paquete ***snort***, concretamente ***será preciso instalar un juego de librerías llamadas* libnet y compilar de nuevo el código fuente de snort con opciones que no están previamente contempladas**.

Antes de explicarte como se hace eso, veamos sus características:

Respuestas en tiempo real (resp: sesp: y react:)

Estas opciones permiten entre otras cosas desconectar la sesión o enviar mensajes ICMP al cliente.

Uso:

Cualquiera de las opciones reseñadas (***resp, sesp, react***) pueden reaccionar de estos modos:

Opción de respuesta	Descripción
Rst_all	Cancela la transmisión o recepción del protocolo TCP
Rst_rcv	Idem, pero solo la recepción
Rst_send	Idem, pero solo la transmisión
Strings:icmp_all	Cancela la transmisión o recepción de mensajes ICMP
Icmp_host	Envía al origen un mensaje del tipo "host inalcanzable"
Icmp_net	Envía un mensaje del tipo "Red inalcanzable"
Icmp_port	Envía un mensaje del tipo "Puerto inalcanzable"

Ejemplos:

alert icmp any any -> any any (resp:icmp_host; msg: "Entrada de un Ping";)

Explicaciones

Esta alerta generará un registro cuando cualquier equipo de la red intente enviar un ping a cualquier destino, cancelará las respuestas del ping y enviará un resultado al origen del tipo "*Host inalcanzable*", *vamos que prohibimos los pings y registramos sus intentos...*

Hay que tener en cuanto **algo importante**... si usamos este tipo de reglas indebidamente podremos causar una negación de servicios (DoS) en la red o bucles entre las sesiones que se establezcan entre cliente y servidor, úsalas con cautela.

OPCIONES ESPECÍFICAS PARA PROTOCOLOS

En conjunción con cualquiera de las reglas explicadas anteriormente, podemos utilizar otras que van en consonancia con el protocolo definido en la alerta...

Son muchas y variadas, algunas bastante útiles y otras menos usadas, si dispones de unos conocimientos medios de TCP/IP las entenderás a la perfección, en caso contrario te crearán muchas dudas, para resolverlas.... ya sabes:

1º) Sigue el curso de TCP/IP que está publicando la revista desde el número 17

2º) Pásate por los foros de hackxcrack para resolver tus dudas

3º) Repasa el Taller de TCP/IP que celebramos en los foros...

En la página oficial de ***snort*** puedes encontrar sus sintaxis y usos,

http://www.snort.org/docs/snort_manual/node15.html

Las opciones específicas para protocolos se basan en las **opciones, cabeceras y formatos de los mensajes TCP, ICMP e IP**

Podemos controlar y analizar: ***Bit de no fragmentación, Tipo de servicio, Source routing, Tamaño de la ventana TCP, peticiones eco-reply, Flags TCP (ACK, PSH, RST, FIN, SYN...) , solicitudes timestamp, tiempo de vida (TTL), números de secuencia y asentimiento de los paquetes...*** lo dicho, es necesario un profundo conocimiento de TCP/IP,

En los foros de hackxcrack (www.hackxcrack.com) colgaré unos documentos para que las puedas entender y aplicar, precisarían de toda una revista como esta para poder explicarlas medianamente bien.

Como muestra pongamos unos cuantos ejemplos:

alert ip any any -> any any (ttl:>25;)
alert tcp $EXTERNAL_NET any -> $ROUTER any (msg:"Router Cisco"; tos:!"0";)
alert tcp any any -> any any (msg: "Sesión establecida";flags:A;)
alert tcp any any -> any any (msg: "Sesión cerrada";flags:FR;)
alert icmp any any -> any any (msg: "respuesta de un PING"; itype:8)
alert icmp any any -> any any (msg: "Petición de un PING"; itype:0)

CLASIFICACION, REVISION y REFERENCIAS.

Las opciones que forman parte de la construcción de reglas que figuran a continuación nos permitirán documentar, establecer prioridades, clasificar las intrusiones, etc.

Una tarea importante a la hora de configurar un IDS **es disponer de una buena documentación** de las vulnerabilidades, bugs, exploits, etc. Si bien no son del todo indispensables, sí son útiles para disponer de más información de las intrusiones y/o conocer donde recurrir para averiguar más datos.

Opciones de Identificación. SID

Se trata de un método para catalogar, distinguir e identificar una regla específica.

Uso: *sid: número*

Dónde el número puede ser:

Menor a 100, Reservados

De 100 a un millón, Utilizados por las distribuciones de reglas de snort

Mayor a un millón: Reglas personales o de libre distribución

Ejemplos:

alert tcp any any -> any 23 (tag:host,10, packets;sid:1200555;)

Número de Revisión (rev:)

Versión de la regla, ya sabes... cuando se modifica una regla original o bien haya cambiado la sintaxis del motor de reglas, podemos incluir la revisión a la que pertenece.

Uso: *rev: versión*

Ejemplos:

alert tcp any any -> any 23 (tag:host,10, packets;rev:2)

Riesgo o Prioridad (priority:)

Identifica el nivel de riesgo o prioridad en la que debe tratarse una regla

Uso: *priority: valor*

Ejemplos:

alert tcp any any -> any 23 (tag:host,10, packets;priority:1;)

Clasificación (classtype:)

La clasificación permite catalogar y agrupar cada regla según el tipo de ataque recibido y se corresponden con la prioridad descrita anteriormente.

Uso: *classtype: nombre de clasificación*

Ejemplos:

alert tcp any any -> any 23 (tag:host,10, packets;priority 3;classtype:protocol-command-decode;)

Explicaciones

Las categorías y prioridades son muchas, te pondré las más comunes:

CLASIFICACION DE ATAQUES DE PRIORIDAD 1	
Attempted-admin	Intento de elevar privilegios como admin.
Attempted-user	Intento de ganar acceso como usuario
Shellcode-detect	Intento de ejecutar una shellcode
Successful-admin	Acceso como admin con éxito
Successful-user	Acceso como usuario con éxito
Troyan-activity	Troyano detectado en la red
Unsuccessful-user	Acceso como usuario fallido
Web-application-attack	Ataque a aplicaciones o servidores Web

CLASIFICACION DE ATAQUES DE PRIORIDAD 2	
Attempted-dos	Intento de provocar un DoS
Attempted-recon	Intento de robar información
Bad-unknown	Tráfico dañino
Denial-of-service	Detección de un DoS
Misc-attack	Ataques varios
Non-standard-protocol	Acceso por protocolo no estándar
Rpc-portmap-decode	Consulta RPC
Successful-dos	DoS con éxito
Successful-recon-largescale	Robo de información a "gran escala"
Successful-recon-limited	Robo de información
Suspicious-filename-detect	Nombre de archivo sospechoso
Suspicious-login	Usuario sospechoso
System-call-detect	Llamada al sistema detectado
Unusual-client-port-connection	Conexiones por puertos no convencionales
Web-application-activity	Acceso a servidores web vulnerables

CLASIFICACION DE ATAQUES DE PRIORIDAD 3	
Icmp-event	Detección de ICMP genérico
Misc-activity	Actividades varias
Network-scan	Escaneo de puertos o redes
Not-suspicious	Tráfico no sospechoso
Protocol-command-decode	Decodificación de protocolo normal
String-detect	Secuencia de caracteres sospechosa
Unknown	Tráfico desconocido

Referencias Externas (reference:)

Esta opción puede ser usada por los plugins de salida para documentar apropiadamente la alerta generada y enlazar con webs externas del tipo bugtraq, securityfocus, etc...

Uso: reference: Sistema, valor

Ejemplos:

alert tcp any any -> any 12345 (reference: CVE, CAN-2002-101; reference:URL, www.securityfocus.com/bid/10456.html; msg: "Troyano";)

A continuación te pongo unos ejemplos de reglas sacados directamente de los archivos de reglas definidos por la distribución oficial de snort, cono todas las explicaciones que hemos dado a cada apartado no deberías tener ningún problema en analizarlas y en comprender lo que hacen:

Ejemplo de alerta para troyano subseven

alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"BACKDOOR subseven 22"; flow:to_server,established; content:"|0d0a5b52504c5d3030320d0a|"; reference:arachnids,485; reference:url,www.hackfix.org/subseven/; classtype:misc-activity; sid:103; rev:5;)

Ejemplo de alerta para negación de servicios en routers cisco

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"DOS Cisco attempt"; flow:to_server,established; content:"|13|"; dsize:1; classtype:web-application-attack; sid:1545; rev:5;)

Ejemplo de alerta para explotar el bug de Unicode en servidores web IIS

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS unicode directory traversal attempt"; flow:to_server,established; content:"/..%c0%af../"; nocase; classtype:web-application-attack; reference:cve,CVE-2000-0884; sid:981; rev:6;)

El orden de las opciones dentro de las alertas no es importante, por convenio se suelen usar las referencias externas, clasificación y revisiones al final, pero no es relevante.

Muchas de estas opciones son usadas por los plug-ins de salida y otras deben ser declaradas como directivas del pre-procesador para que puedan ser incluidas dentro de las reglas.

En el próximo apartado veremos las directivas del pre-procesador, entonces lo comprenderás mejor, pero antes vamos a realizar alguna práctica... que ya va siendo hora.

Práctica 2. Preparar snort para el uso de Contramedidas en Tiempo Real

Hemos hablado de las reglas junto con FlexResp y si recuerdas dije que esta posibilidad no viene "por defecto", para habilitar esta opción (y otras) serán precisas varias acciones:

- Instalar snort desde el código fuente, no desde un RPM ni desde los binarios RPM
- Habilitar las opciones que deseemos implementar vía ./configure
- Instalar las librerías necesarias

Paso 1) Descargar el código fuente de snort

Lo primero descargar las fuentes de snort desde la página oficial:

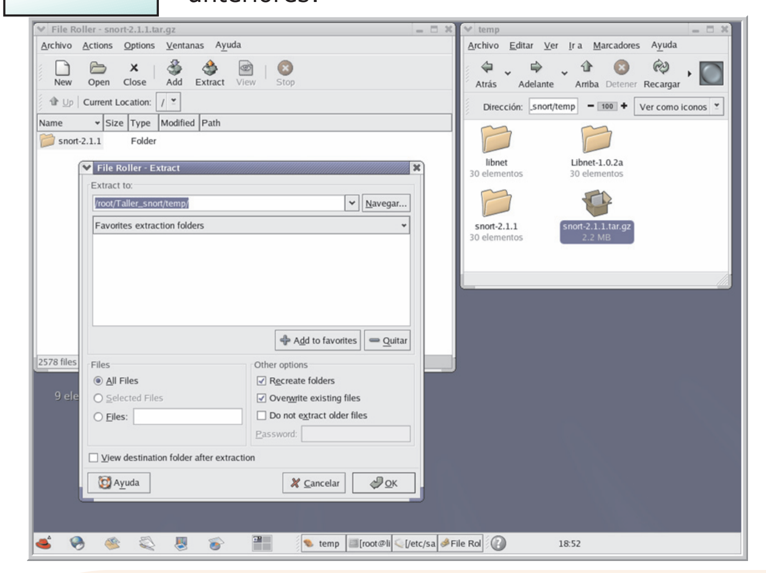
http://www.snort.org/dl/snort-2.1.1.tar.gz

Una vez descargado el código fuente, lo guardamos en un directorio y descomprimimos el tarball en el directorio que queramos, en el ejemplo que se sigue el archivo descargado de snort.org se guardó en /root/Taller_snort/temp y los archivos descomprimidos en /root/Taller_snort/temp/snort-2.1.1

Para descomprimirlo podemos usar la orden tar (como vimos en el primer artículo) y si estamos usando Xwindow, bastará con hacer doble-clic en el archivo snort-2-1.1.tar.gz y extraerlo en la carpeta que queramos.

En la siguiente pantalla se muestran los pasos anteriores:

Pantalla 8. Descomprimir las fuentes de snort en los directorios indicados



Paso 2) Instalar las librerías necesarias

Ya sabemos que para que ***snort*** pueda ejecutarse se precisan las librerías *libpcap*, pero si además queremos utilizar otras funciones como las respuestas flexibles, necesitamos disponer de las librerías ***libnet***

Para ello, primero las bajamos:

http://www.packetfactory.net/libnet/dist/deprecated/libnet-1.0.2a.tar.gz

Existe otra versión *"mas actual"*, la *1.1.x* pero al menos yo no conseguí que funcionase correctamente, recientemente ha aparecido otra versión más moderna de ***snort*** y puede que sí sean operativa con ella, pero no vamos a empezar ahora actualizando ***snort***... lo dejo para ti si quieres probar esa última revisión que es la 2.1.2

Las librerías ***libnet*** también las guardamos en el mismo directorio que antes, para no tener desperdigadas todas las utilidades. Si te fijas en la pantalla que puse anteriormente, el conjunto de **librerías libnet *están /root/Taller_snort/Libnet-1.0.2a***

Y las descomprimimos igual que antes, por ejemplo en la carpeta libnet que cuelga de ***/root/Taller_snort/temp,*** como ilustra la pantalla anterior.

Paso 3º) Instalar las librerías libnet

Esto es sencillo, accedemos al directorio donde está situado el resultado de la descompresión y ejecutamos ***./configure*** desde la línea de comandos de un Terminal.

Luego creamos los ***makefiles*** con la orden ***make***

Y por último instalamos los makefiles con make install, vamos lo de siempre... te lo pongo todo seguidito...

```
cd /root/Taller_snort/temp/libnet
./configure
make
make install
```

Paso 4º) Instalar snort desde las fuentes...

Como antes primero accedemos al directorio donde descomprimimos el archivo ***tarball***, que si recuerdas era ***/root/Taller_snort/temp/snort/snort-2.1.1***

Ahora ejecutamos ***el script ./configure*** pero.... hay que indicarle alguna opción si queremos que todo vaya bien.... entre las opciones que podemos ejecutar están:

./configure --enable-smbalerts, si deseamos usar posteriormente la posibilidad de mandar alertas a equipos *Windows* mediante los mensajes emergentes (***WinPopup***)

./configure --enable-flexresp, precisamente esta es la opción que buscábamos, esto habilitará la posibilidad de **incluir en nuestras reglas las contramedidas en tiempo real**

./configure --with-mysql=DIR, soporte para ***MySQL***, imprescindible si deseamos utilizar plug-ins de salida del tipo ***ACID*** o cualquier registro de alertas en una ***BBDD MySQL***

./configure --with-odbc=DIR, idem de lo anterior, pero para Bases de datos ***ODBC***

./configure --with-postgresql=DIR, más de lo mismo pero para Gestores ***PostGreSQL***

./configure --with-oracle=DIR, lo mismo para ***ORACLE***

./configure --with-openssl=DIR, Soporte para ***SSL***, se usa con plug-ins ***XML***

./configure --with-libpq-includes=DIR, indica el directorio de ***includes*** para soporte de Base de datos ***PostgreSql***

./configure --with-libpq-libraries=DIR, Indica el directorio donde están las librerías necesarias para ***Postgres***

./configure --with-libpcap-includes=DIR, necesario para especificar el directorio en donde están alojadas los ***includes*** de las librerías ***libpcap***, esto no es del todo preciso, sólo si el script no es capaz de encontrar las librerías o si las tenemos guardadas en directorios diferentes al de por defecto.

./configure --with-libpcap-libraries, idem del anterior pero para las librerías

Bien, nuestra línea de órdenes será:

```
cd /root/Taller_snort/temp/snort-2.1.1
./configure –enable-smb-alerts –enable-flexresp –with-mysql
make
make install
```

Observa que **no se indicó el directorio de instalación del Gestor de Bases de Datos MySQL**, los parámetros citados anteriormente sólo necesitan especificar el directorio si el script ***./configure*** no encuentra el programa.

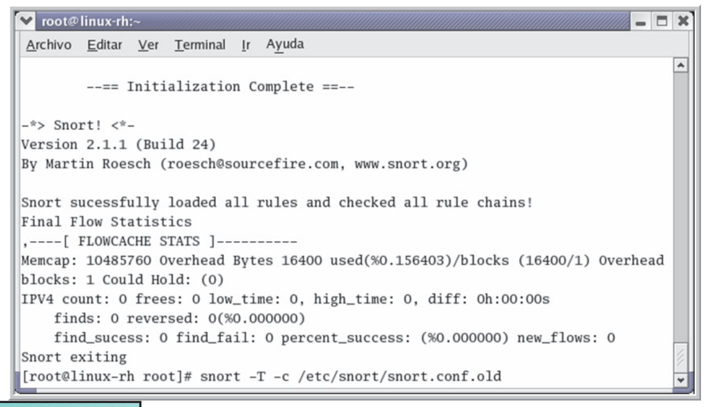
Ni que decir tiene que ***MySQL*** debe estar instalado en nuestra máquina y que debemos contar con el compilador ***gcc*** para que esto se pueda hacer, pero esas dos cosas están prácticamente en todas las distribuciones de ***LiNUX***, si no es el caso, ya sabes... paquetes y visitar la web de www.mysql.com para descargar el gestor de Base de Datos.

Para esta práctica no necesitamos ***smb*** ni ***mysql***, eso será para el próximo artículo... pero así ya llevamos camino adelantado.

Llegados a este punto deberíamos tener ***snort*** preparado.... podemos probarlo para ver si todo es correcto, por ejemplo:

snort –T –c/etc/snort/snort.conf.old

La opción **–T** (Test) comprueba que todo se pueda ejecutar convenientemente y que las reglas se pueden incluir, **recuerda** que si has seguido TODOS los pasos desde el primer artículo, el archivo de configuración original lo renombramos a ***snort.conf.old***, si no es así prueba con el tuyo propio o indica la ruta y archivo de configuración que uses tras la opción –c



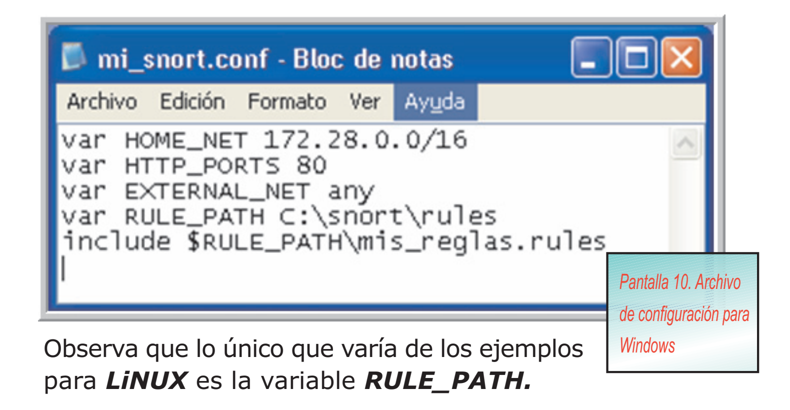
Pantalla 9. Verificación de la instalación de snort con opciones especiales

Te estarás preguntando... ***¿Y qué fue de nuestro querido Windows?***

Pues nada... Si instalaste la versión de ***snort*** para *Windows*, **todo esto no es necesario** (dejando a un lado que no es posible). Lo que quiero decir es que si deseamos implementar las opciones de contramedidas en tiempo real dentro de las reglas de snort, **con la versión Windows ya se instalan las librerías necesarias, etc...,** *algo bueno debería tener, no?*

Vamos que lo único que deberíamos tener en cuenta es que la estructura de directorios no es la misma que en *LiNUX*, siempre digo que no lo repetiré más... y siempre caigo en los mismo... repito por "*penúltima*" vez, pero más breve...

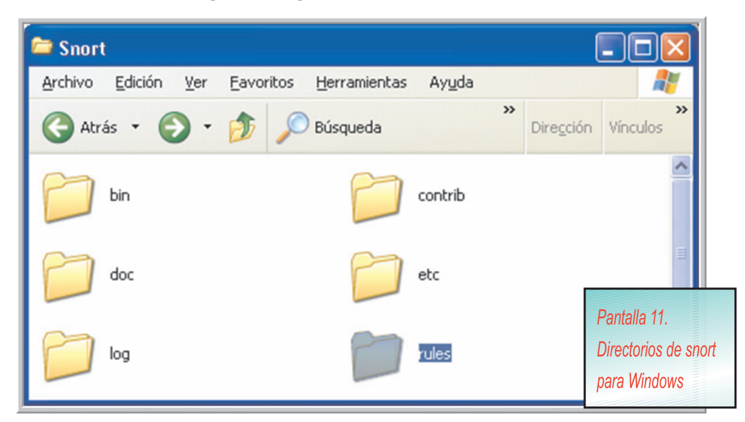
El archivo de configuración en C:\snort\etc y modificamos el contenido, así:



Pantalla 10. Archivo de configuración para Windows

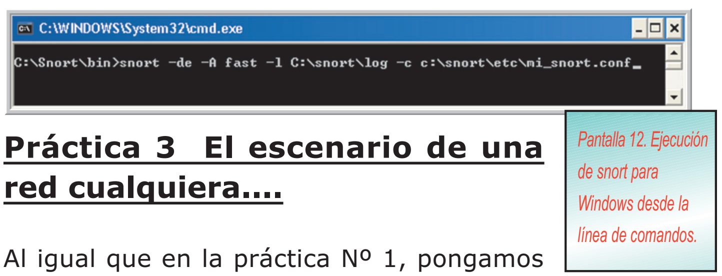
Observa que lo único que varía de los ejemplos para ***LiNUX*** es la variable ***RULE_PATH.***

Los **logs** en **C:\snort\logs** y Las **reglas** en el directorio **C:\snort\rules**



Pantalla 11. Directorios de snort para Windows

Por tanto la orden sería:



Pantalla 12. Ejecución de snort para Windows desde la línea de comandos.

Práctica 3 El escenario de una red cualquiera....

Al igual que en la práctica Nº 1, pongamos un escenario probable y unas políticas para ***snort***.

1º) **Los directivos** de la empresa tienen equipos con **IP's: 172.28.0.50 y 172.28.0.99**, nuestra intención es **que todo aquel que intente hacer un ping a estos equipos reciba un mensaje del tipo "Host Inalcanzable**" pero ellos SI DEBEN poder hacer ping a los demás.

2º) Prevenir que **ningún cliente de la red acceda a internet si se detecta la palabra "sex"** en la dirección Web.

3º) Disponemos de una **red de laboratorio** (de pruebas) y queremos que esos equipos **sólo puedan comunicarse entre ellos** y que bajo ningún concepto puedan acceder a recursos externos o internos en nuestra misma red, sabiendo que estos equipos de laboratorio son **172.28.0.10, 172.28.0.11 y 172.28.0.12**

4º) **El router** de la empresa que brinda conectividad a otras redes es la IP **172.28.0.1** y **se puede administrar vía telnet por el puerto 23 y/o http por el puerto 8080**, queremos que **SOLO el administrador de la red pueda pasar llegar al mismo** y que se disparen las alarmas si el login falla, el admin. **Es uno de los JEFES con IP 172.28.0.50**

....

....

El ejemplo se basa en un ***IDS corriendo en LiNUX***, recuerda los cambios necesarios para Windows en la estructura de directorios y variable ***RULE_PATH***

El archivo de configuración a crear será ***mi_empresa.conf***

El archivo con estas ***reglas*** específicas será: ***mi_empresa.rules***

El directorio de logs será: ***/var/logsmi_empresa***

Paso 1º) Crear un nuevo archivo de configuración

```
var RED_EMPRESA 172.28.0.0/16
var SERVIDOR_WEB 172.28.0.24
var JEFES [172.28.0.50,172.28.0.99]
var ROUTER 172.28.0.1
var RED_PRUEBAS [172.28.0.10,172.28.0.11,172.28.0.12]
var EQUIPO_ADMIN 172.28.0.50
var RED_EXTERNA any
var RULE_PATH /etc/snort/rules
include $RULE_PATH/mi_empresa.rules
```

Lo guardamos en el directorio ***/etc/snort*** con el nombre ***mi_empresa.conf***

IMPORTANTE: NO dejes espacios entre IP e IP después de las comas en la declaración de las variables ***JEFES y RED_PRUEBAS*** o no funcionará....

Paso 2º) Creamos las reglas en el archivo *mi_empresa.rules*

```
alert icmp any any -> $JEFES any
(resp:icmp_host;itype:8;msg: “Ping a los
JEFES denegado”;)

alert tcp any any -> $RED_EXTERNA 80
(resp:rst_all; content: “sex”; msg: “Webs
de SEXO no permitidas”;nocase;)

alert tcp $RED_PRUEBAS any ->
!$RED_PRUEBAS any (resp:rst_all; msg:
“Equipo de Laboratorio fuera de RED
PERMITIDA”;)

alert udp $RED_PRUEBAS any ->
!$RED_PRUEBAS any (resp:rst_all; msg:
“Equipo de Laboratorio fuera de RED
PERMITIDA”;)

alert icmp $RED_PRUEBAS any ->
!$RED_PRUEBAS any (resp:rst_all; msg:
“Equipo de Laboratorio fuera de RED
PERMITIDA”;)

alert tcp !$EQUIPO_ADMIN any ->
$ROUTER 8080 (resp:rst_all; msg: “Login
al Router por HTTP no permitido”;)

alert tcp !$EQUIPO_ADMIN any ->
$ROUTER 23 (resp:rst_all; msg: “Login
al Router por TELNET no permitido”;)

alert tcp $EQUIPO_ADMIN any ->
$ROUTER 23 (content: “login failed”;
session: printable; nocase;)
```

Alguna aclaración es necesaria...

En la primera regla, es NECESARIA la **opción itype:8**, *¿por qué?* Pues veamos, cuando se realiza un ping (petición eco) se recibe un pong (eco-reply). Si no ponemos *iptype:8*, se registrarán los eco-reply que los propios equipos de los jefes realizan (*falso positivo*), es decir, los jefes no podrán hacer ping ni tampoco a ellos puesto que las ip's de los jefes también pertenecen a la red de la empresa, o tampoco podrían hacerse ping entre ellos.

Sin embargo en la **regla icmp** que niega los pings desde la ***RED_PRUEBAS*** hacia cualquier destino que NO SEA (!) la ***RED_PRUEBAS*** no es preciso, porque precisamente lo que queremos es negar TODO.

Si observas la declaración de reglas, más concretamente **en la que "filtra" y deniega el acceso a páginas de sexo...** en ***content*** puse ***sex*** y no sexo... esto trae alguna anécdota... *imagina que un usuario de la red abre google y busca Canciones Camilo Sexto... jeje, como imagino estarás pensando, cortará la comunicación y registrará un alerta.... (Camilo **Sex**to) es decir, todo aquello que tenga sex será "baneado", claro que quien demonios buscará esas canciones... perdón a los amantes del Sr. Sexto...*

Observa también que las ***reglas en que aparece !$EQUIPO_ADMIN***, con esto nos aseguramos que sean TODOS excepto el admin.....

Bueno hay muchas formas de aplicarlas y construirlas, seguro que se te ocurren más y con otras variantes, de eso se trata... de ir "*personalizando*" el archivo de configuración según las necesidades de cada uno...

Lo guardamos dentro del directorio ***/etc/snort/rules*** con el nombre propuesto: ***mi_empresa.rules***

Paso 3º) Crear el directorio de logs dentro de /var

mkdir /var/logsmi_empresa

Paso 4º) Ejecutamos snort desde la línea de comandos así:

snort –de –A: fast –l /var/logsmi_empresa –c /etc/snort/mi_empresa.conf

Y como ya hemos visto en otras ocasiones se registrarán las acciones expuestas en el escenario de la práctica.



IMPORTANTE

IMPORTANTE: Haz pruebas, revisa los logs, las alertas, etc... Ahh!! La gente de Windows... En equipos *Windows* el archivo contenedor de las alertas no se llama alert, se llama alert.ids (una pequeña variación, pero que habrá que tenerla MUY en cuenta cuando usemos *plugins* de salida)

Tampoco hemos usado todas las opciones entre otras cosas porque para alguna de ellas es necesario configurar el pre-procesador... cosa que no hemos aprendido...

NO TODAVÍA, eso es lo que tenemos por delante AHORA MISMO

EL PREPROCESADOR

A estas alturas ya sabemos lo que es y para lo que sirve el *pre-procesador*, tratemos de explicar un poquito más sus características y sus componentes.

Cuando un paquete se mueve por la red y ***snort*** lo captura, antes de aplicar las reglas pasa por el *pre-procesador* y, en su caso, se aplican las directivas especificadas en el mismo.

El *pre-procesador* está concebido como un conjunto de *plug-ins* (que se pueden o no incluir) para analizar el tráfico de datos en la red, estos *plug-ins* **se declaran en el archivo de configuración de snort.**

Entre las directivas del *pre-procesador o plug-ins* más comunes tenemos:

Reconstrucción o reensamblaje de paquetes: ***Stream4 y frag2***

Decodificación de protocolos: ***Telnet, http, RPC, etc..***

Exploración de puertos: ***portscan y portscan2***

Experimentales: ***arpspoof, asn1, shellcodes***

Además, alguno de estos pre-procesadores son necesarios cuando se utilizan plug-ins de salida, pero como vimos hasta ahora, ***snort*** puede ejecutarse sin necesidad de ellos.

Prepocesador stream4

Se basa en dos conceptos: ***TCP statefulness y Session reassembly*** (Reconstrucción de conexiones completas o incompletas del protocolo TCP y reensamblado de la sesión).

Para entender ***Statefulness*** es necesario hablar "*un poquito*" del protocolo TCP, más concretamente de **cómo se establecen las conexiones TCP**, **el saludo de tres vías y las señales TCP o flags:**

TCP y el Saludo de tres vías

Cuando dos equipos se comunican mediante TCP, ocurre algo así:

1º) **Sincronización** (***SYN***) que envía el emisor al receptor

2º) **Respuesta *SYN-ACK***, que envía el receptor al emisor

3º) **Envío *ACK*** del emisor al receptor

De esta forma emisor y receptor se "*ponen de acuerdo*" para el envío y recepción del mensajes, se envían los *socket* correspondientes, los números de puerto, la secuencia....

Es como cuando llamas al novio o a la novia por teléfono...

*1º) Marcas el número y suena la llamada en el teléfono destino (**SYN**)*

*2º) El destino descuelga (acepta la llamada) ¿Diga? ¿Quién es? (**SYN+ACK**)*

*3º) Hola cariño.... soy yo.... (**ACK**)*

Es algo más complicado que eso... realmente intervienen otros "*actores*" en la película... los números de secuencia y asentimiento, ***en el número 14 de la revista*** tienes un artículo sobre ***hijacking*** que explica más detenidamente el proceso en la negociación del protocolo TCP, si no dispones del mismo lo puedes descargar de aquí:

http://www.forohxc.com/hijacking/articulo14/hijacking.pdf

La razón por la que *TCP* negocia las conexiones es precisamente porque:

- *TCP* es un protocolo orientado a conexión (si no se completa el mecanismo del saludo de tres vías no se establece una comunicación efectiva).
- Detecta errores si los paquetes se pierden (se realiza un seguimiento de cada transmisión y si uno de los extremos no recibe o recibe mal, se pide de nuevo el paquete con número de secuencia perdido).
- Se abren puertos en el lado del cliente y del servidor para mantener "*viva*" la comunicación.
- Se termina de forma educada (señal *FIN*) o por "*las bravas*" (señal *RST*).

Además, tanto el cliente como el servidor no se comunican "*byte por byte*" o paquete por paquete, negocian un tamaño y cantidad de datos de antemano, a esto se llama ***Ventana TCP (TCP Window)*** que no es otra cosa que la **cantidad máxima de información que**

un extremo puede absorber o enviar sin que haya recibido del otro lado una señal ACK.

En condiciones normales esta es una buena técnica, ordenada, metódica y correcta... pero cuando vienen "*los chicos malos*" TCP puede convertirse en un problema.

Podemos usar TCP para *escanear* puertos y servicios a la escucha de servidores, para "*falsear*" una comunicación, para secuestrarla, para causar una negación de servicios, para enviar paquetes "*mal formados*" y provocar respuestas erróneas de los servidores... vamos que en un mundo inseguro como es la red, no podemos dejar el trabajo a TCP en lo que se refiere a seguridad y privacidad.

Y no bastará un simple *firewall*, porque si necesitamos ofrecer un servicio de cara a las redes externas (como puede ser un servicio web, una tienda electrónica, correo, etc...) nuestro *firewall* no debe bloquear esos puertos o servicios, si lo hacemos así no podremos ofrecernos como servidores de nada.

El ***pre-procesador stream4*** de *snort* está pensado y construido para analizar el tráfico *TCP* y evitar acciones poco habituales, como pueden ser exploración de puertos o redes completas, negociaciones anómalas del protocolo, señales o flags mal intencionadas, evitar la fuga de paquetes y otras directivas que nos permitirán optimizar *snort* en el consumo de recursos (RAM), conexiones pendientes, límites en la conexión y desconexión, etc...

Permite hacer un seguimiento de múltiples sesiones... hasta 65536 simultáneas y **será de obligado uso si en nuestras reglas incluimos opciones de control de flujo, monitoreo de paquetes, tiempos de conexión, etc.** Dicho de otro modo, si queremos utilizar reglas de control de flujo que vimos antes deberemos incluir las directivas del ***pre-procesador stream4.***

Configurando stream4

En el archivo de configuración debemos incluir lo siguiente:

preprocessor stream4

¿Ya? Pues sí… con eso sería suficiente... sin embargo podemos realizar "*un ajuste fino*" si incluimos algunas opciones características del *preprocesador*

Componentes y parámetros de configuración de stream4

detect_scans: por defecto está **desactivado** y como puedes ir suponiendo advertirá o no de los intentos de exploración de puertos, para ello utiliza y examina las señales ***ACK, FIN, RST, SYN, PSH, URG,*** etc... del protocolo ***TCP***.

preprocessor stream4: detect_scans

detect_state_problems: Por defecto **desactivado**, detecta abusos en la negociación del protocolo TCP, como tamaños de ventana excesivamente pequeños o grandes, alteraciones en el número de secuencia y asentimiento, etc. Al activar esta directiva podemos encontrarnos con "*falsos positivos*", cada sistema operativo tiene su propia idiosincrasia en la arquitectura de la *Pila TCP/IP* y puede ocurrir que salten alertas en comunicaciones que no deberían ser objeto de sospecha.

preprocessor stream4: detect_state_problems

disable_evasion_alerts: También está **desactivado** por defecto, esta directiva permite reconstruir los paquetes y chequear los intentos de falsificación de los mismos o los intentos de sobrescribir los datos de uno ya recibido, esto es una técnica muy habitual en los ataques *DoS*, sin embargo, en una comunicación normal, también ocurren este tipo de circunstancias, los paquetes se pierden por múltiples motivos y la comunicación se

reinicia o cambian los números de secuencia por lo que en ocasiones también pueden producirse falsos positivos si esos casos se producen con demasiada frecuencia. Lo mejor es no incluirla, recuerda que ***disable*** es desactivar, de tal modo que ***snort*** estará "*al loro*" si se producen "*malas artes*" en nuestra red... claro que siempre se puede incluir para "*vigilar*" a quien se cree "*mas listo*"... empezamos a entender ahora lo que es un ***honeypot***... un señuelo, un equipo aparentemente indefenso....

preprocessor stream4: disable_evasion_alerts

ttl_limit, este parámetro marcará el tiempo máximo tolerado entre el envío y recepción de paquetes dentro de una misma sesión. Habitualmente el número de *routers*, saltos, etc entre el origen y el destino es el mismo que entre el destino y el origen (*por donde voy, vengo*) pero tampoco tiene por qué ser así, y de eso se ocupa este parámetro.

Si el número de saltos de vuelta en el paquete es excesivo con respecto al número de saltos que sufrió en la ida*... algo pasa....* a lo peor "*alguien*" secuestró la conexión o hizo que nuestros datos se enrutaran de forma no convencional hacia un destino intermedio.... Es difícil indicar un número mágico, dependerá de los protocolos de enrutamiento que usen nuestras redes y a las que nos conectamos, pero como orientación, no debería existir una diferencia de más de 10 saltos, es decir, si en "*la ida*" hacia el destino saltamos por 6 *routers*, *"a la vuelta"* hacia el origen no se debería de sobrepasar de 16.

preprocessor stream4: ttl_limit 10

keepstats*:** almacena estadísticas por cada sesión por cada máquina o en formato binario (machine o binary***) para que puedan ser usados por los *plugins* de salida apropiados.

preprocessor stream4: keepstats binary

noinspect, por defecto desactivada... *faltaría mas*... esto desactiva la inspección de paquetes

preprocessor stream4: noinspect

timeout: Por defecto a 30 segundos, es el tiempo en el que stream4 dejará de atender una conexión si no hay actividad en la misma...

preprocessor stream4: timeout 25

memcap: es el número máximo de memoria expresada en bytes que utilizará stream4 para reensamblar los paquetes, por defecto unos 8 megas, es conveniente asignar más si nos lo podemos permitir, revisa los procesos que corren en tu máquina para saber la RAM que te queda disponible y ajustar mejor este parámetro.

preprocessor stream4: memcap 65000000

stream4_reassemble: es un *plugin* específico para la reconstrucción de los paquetes TCP por puertos, puede tener varias opciones: ***clientonly, serveronly, both, noalertsy ports***. Y reconstruirá el tráfico por los puertos indicados (***ports*** lista de puertos...) atendiendo a la conexión establecida por el servidor, el cliente o ambos (***clientonly, serveronly, both***) y generando alertas o no generándolas (***noalerts***)

preprocessor stream4_reassemble both ports 21 23 25 53 80 138 139 445

Ni que decir tiene que estas directivas pueden incluirse en una sola o varias líneas, Ejemplo:

preprocessor stream4: ttl_limit 10
preprocessor stream4: detect_scans, detect_state_problems, keepstats

Fragmentación de paquetes. Preprocesador frag2

En el artículo anterior ya hablamos de ello, la fragmentación es necesaria pero como todo hay que controlarlo y un excesivo número de

paquetes muy fragmentados pueden tirar abajo al sistema más pintado.... Son ataques muy comunes en la negación de servicios y pretenden mantener tan ocupado al sistema que termina por bloquearse... amén de algunos ataques mas sibilinos que sobrescriben unos fragmentos con otros y pueden provocar un desbordamiento de memoria y terminar por entregarnos una *shell* del sistema (o en algunos casos matando al servidor).

Por ello y para ello existe ***frag2***, pero no te fíes... no vaya a ser que te maten el *IDS*

Configurando frag2

Al igual que *stream4*, las directivas de ***frag2*** deben indicarse en el archivo de configuración:

preprocessor frag2: timeout 60 memcap 6000000

Algunas de las opciones disponibles son bastante similares a stream4, pero aplicadas a la fragmentación de paquetes en lugar de a la reconstrucción y fluido de los mismos, solo me extenderé en aquellas nuevas o en sus variaciones:

timeout, por defecto 30 segundos y con el mismo objetivo que en *stream4*

memcap, por defecto a 4 megabytes

ttl_limit, idem de *stream4*

min_ttl, es el tiempo mínimo necesario para permitir que ***frag2*** reconstruya los fragmentos y previene a ***frag2*** de aquellos paquetes que no pudieron llegar a su destino.

detect_state_problems, aplicado a la fragmentación... los problemas pueden ser demasiados fragmentos, desplazamientos u *offset* aleatorios y extraños, sobre-escritura de paquetes.

Prepocesadores para la decodificación de protocolos

Después de lo que nos ha llovido, esto no tiene mayor misterio

Decodificación TELNET

preprocessor telnet_decode: 23 25 80

Pues eso, decodifica el protocolo *telnet* que pueda ser susceptible de utilizarse por la lista de puertos que le sigue

Decodificación y opciones http

http_inspect es un decodificador genérico para el protocolo HTTP, analiza cabeceras, tanto por parte del cliente (el navegador) como por parte del servidor (El servidor web)

Hay dos secciones o áreas a configurar: ***Global*** para el entorno global de los servidores Web y ***Server*** para definir directivas específicas de un servidor web en concreto.

Configuración Global de *http_inspect*

Determina el funcionamiento global (de ahí su nombre) de las funciones del pre-procesador para cualquier servidor web y acepta las siguientes opciones:

```
preprocessor http_inspect: global \
        iis_unicode_map <map_filename>\
        codemap <integer> \
        [detect_anomalous_servers]\
        [proxy_alert]
```

global: es de obligatorio uso y sólo debe existir una definición global en el archivo de configuración

iis_unicode_map <map_file>: También es de uso obligatorio o podemos obtener u error, debemos indicar un archivo que "*apunta*" la codificación en formato *unicode* de los

caracteres para los diferentes idiomas, el archivo por defecto es ***unicode.map***, este archivo lo encontrarás en el directorio /etc de la instalación de ***snort***, es un archivo de texto que puedes modificar

codemap <integer> hace referencia al juego de caracteres usado por unicode, por ejemplo si buscas dentro del archivo ***unicode.map***, encontrarás que el número *1252* corresponde al juego de caracteres ***ANSI - Latin I***

Dentro de la carpeta */contrib*, hay un fuente en C que se llama ***ms_unicode_generator.c*** que te permitirá crear tus propios archivos y juegos de caracteres, aunque para nosotros utilizar el 1252 es el más conveniente.

detect_anomalous_servers: habilita la inspección del tráfico HTTP sobre puertos no asignados a dicho protocolo, es decir, que alertará si se usa http por puertos que no estén designados como el puerto en el que escucha el servidor web, normalmente el 80.

Ten cuidado al usar esta directiva y no la uses si no se definió un servidor web por defecto o causará un gran impacto en el trabajo de ***snort***.

proxy_alert: habilita alertas globales en el uso de *proxys web*, es decir, en combinación con otra directiva dentro de la sección *server* (***allow_proxy_use***) recibirás alertas de los usuarios que no utilicen la configuración del *proxy web* predeterminada y podrás obtener registros de aquellos usuarios de la red que salen por *proxys webs* que no estén definidos por el administrador.

Ejemplo de uso:

preprocessor http_inspect: global iis_unicode_map unicode.map 1252

Configuración ***server***

Server: que podrá ser default o una dirección IP

Profile: podemos especificar si son todos (all) o uno específico (iis ó apache)

Ports: se indicarán entre llaves el puerto o puertos por los que escucha el servidor web

Ejemplos:

La siguiente directiva especifica un servidor por defecto (ya sea *apache* o *IIS*) que escucha por los puertos 80, 8080 y 8000

preprocessor http_inspect_server: default profile all ports {80 8080 8000}

La siguiente directiva es la misma pero para el servidor web instalado en la dirección IP 172.28.0.99 configurado por le puerto 80

preprocessor http_inspect_server: server 172.28.0.99 profile all ports {80}

Además podemos incluirle algunas opciones, entre las más significativas están:

flow_depth <valor>

Indica el número de *bytes* que inspeccionará ***snort*** como *payload*, o sea, los *bytes* que se analizarán dentro del protocolo http, los mejores valores estan entre 150 y 300, valores muy grandes pueden afectar al rendimiento de ***snort*** perjudicialmente.

u_encode <yes|no>

Se utiliza para traducir o codificar determinados caracteres *ASCII* en formato hexadecimal, en 32 bits que es lo que maneja ***unicode*** en lugar de los 8 bits que usa ASCII, *bueeeenoooo, son 7 pero para no extenderme en ello.....*

Imagino que ya conoces los problemas que la traducción de ***unicode*** reportó a los ***IIS de Microsoft...*** *y es alucinante, pero sigue causando estragos, en* ***routers CISCO*** *(lider de comunicaciones en Internet) continúan los problemas de acceso por* ***unicode****...*

iis_unicode <yes|no>

Si por ejemplo ponemos en ***formato unicode %af%90*** *esto mismo* ***se puede representar por %uAF90****, ¿no ,lo sabías? Pues agárrate si tienes un IIS como servidor web... puede seguir siendo vulnerable a* ***unicode*** *si le mandamos una url en ese formato....*

double_decode <yes|no>

Seguimos con la saga... esto es la "*doble decodificación*", de ello se ha hablado y se sigue hablando... ***imprescindible si tienes un IIS en la red***... y otros... no te creas que es el único, aunque lo tengas parcheado... usa esta opción.

multi_slash <yes|no>

Esta opción normaliza el uso de múltiples barras en la dirección web, es decir, direcciones del tipo ***////////direweb*** se normalizará a ***/direweb***

iis_backslash <yes|no>

Esto le indica que "*traduzca*" **las barras \ por barras /,** muchos ataques y vulnerabilidades de correo, web, etc... se producen por el uso "*no uniforme*" de las barras en las direcciones web.

directory <yes|no>

Parecida a la anterior, normaliza el acceso transversal a directorios del tipo:

/foo/fake_dir/../bar *se normaliza por:* ***/foo/bar***

o también,

/foo/./bar *queda como:* ***/foo/bar***

apache_whitespace <yes|no>

Elimina los espacios "*sobrantes*" o ajusta a un espacio la decodificación del protocolo.

Puedes encontrar otras opciones de interés aquí:

http://www.snort.org/docs/snort_manual/node17.html

Entre ellas: ***iis_delimiter <yes|no> chunk_length <valor mayor a cero> no_pipeline_req non_strict allow_proxy_use oversize_dir_length <valor mayor a cero> inspect_uri_only***

Ejemplos

preprocessor http_inspect_server: server 172.28.0.99 ports { 80 } flow_depth 0 double_decode yes

preprocessor http_inspect_server: server default ports { 80 } flow_depth 300 apache_whitespace yes directory no iis_backslash no u_encode yes double_decode yes iis_unicode yes iis_delimiter yes multi_slash no

preprocessor http_inspect_server: server default profile all ports { 80 8080 }

Configurando rpc_decode

Seguro que más de una vez oíste hablar del ***protocolo RPC***, últimamente se puso de moda con el famoso *virus blaster* y los equipos *Windows*, pero no es exclusivo de *Windows*, RPC es un protocolo usado en casi todas las redes, lo usan servicios como NFS, hasta DHCP y muchos otros servicios de red.

RPC no es un protocolo "*normal*", no es un protocolo de transporte de datos, se sitúa en la ***capa de sesión del modelo OSI***, **utiliza tanto UDP como TCP** y básicamente **se utiliza para permitir un mecanismo de "*abstracción*" en el que un *host* llama a un programa que está corriendo en otro *host*.**

RPC también utiliza puertos, pero a diferencia de protocolos como *telnet* o *ftp*, un servicio RPC basado en un ***servidor RPC*** puede lanzarse sobre puertos no reservados, esto es lo que se llama ***rpcbind o portmapper***.

LiNUX utiliza puertos *TCP y UDP 111*, mientras que *Windows* usa el *135* y otros sistemas operativos pueden utilizar el servicio RPC por puertos diferentes, **snort comprueba RPC entre los puertos 111 y 32771** mediante

el ***plug-in rpc_decode*** que serán indicados en la lista de puertos al definir la directiva.

Las opciones más comunes en ***rpc_decode*** son:

alert_fragments, por defecto desactivado y alertará si se producen excesivos fragmentos dentro de los registros que maneja el servicio.

no_alert_múltiple_request, RPC se basa en sesiones, normalmente esas sesiones son de "*dos o tres vías*" con más de una petición por paquete, esta opción no alertará si se produce una negociación del protocolo si en un paquete se incluye una única petición,

no_alert_large_fragments, mejor no usarlo, puesto que muchos de los desbordamientos de buffer y *overflows* se producen mediante esta técnica.
no_alert_incomplete, como los anteriores, también está desactivado por defecto, y como es de suponer también es causa de múltiples vulnerabilidades.

Ejemplo:

preprocessor rpc_decode: 111 135 32771

Configurando portscan y portscan2

No creo que sea necesario decir lo que es una exploración de puertos ni para lo que se utiliza... bueno por si no lo sabes, *es como ir llamando a las puertas de una casa para haber si hay alguien dentro..* es decir, un ***escaneo*** de puertos **pretende conocer los servicios que ofrece un host remoto.**

Portscan2 es un *plug-in experimental*, bueno ya no tanto, de hecho está pensado para sustituir a su "*hermano pequeño*" ***portscan***.

Ambos son muy fáciles de usar y entender,

Para usar ***portscan*** basta con: ***la red, nº puertos, el tiempo de detección y un fichero***

Ejemplo:

Prepocessor portscan: 172.28.0.0/16 5 30 /var/log/escaner.log

Esta orden creará un **registro en el archivo escaner.log si en la red 172.28.0.0 /16 cuando se detecten más de 5 puertos escaneados en 30 segundos.**

Otra forma de *portscan* es:

Preprocessor portscan-ignorehosts: 172.28.0.50/16 172.28.0.200/16

Que excluirá los *escaneos* efectuados a las máquinas que se incluyen en la lista de *hosts*

Los escaneos que detecta ***portscan*** son:

FULL, cuando se establece una conexión completa en el saludo de tres vías de TCP
NULL, cuando todos los flags TCP están desactivados
FIN, cuando sólo se activa el flag FIN
SYN-FIN, Flags SYN y FUN activadas
XMAS, FIN, SYN, URG y PSH están activadas

Son varios los números de la revista donde se explicaron estos diferentes modos de escaneo, busca las revistas que ya se han publicado y hablan de ***nmap (revista 13)***

Las opciones que podemos usar con ***portscan2*** son:

target_max, por defecto a 1000, que es el número máximo de objetivos y seguimientos que guardará portscan2 para analizar

scanners_max, también 1000 por defecto, es el máximo número de IP's que registrará.

target_limit, por defecto 5,

port_limit, por defecto a 20,

timeout, por defecto a 60 segundos,

log, por defecto al es el archivo /scan.log,

Ejemplos:

preprocessor portscan2: target_max 1000, scanners_max 1000, port_limit 20

Y también sirve la exclusión de *host* como antes:

preprocessor portscan2-ignorehosts: 172.28.0.50/16 172.28.0.200/16

Para terminar vamos a describir otros tres *pre-procesadores* en "*experimentación*" pero bastante potentes... son: ***conversartion, arpspoof y fnord***

Configurar el pre-procesador conversation

Registra "*las conversaciones*" entre dos *host* que usan protocolos no orientados a conexión como UDP, las opciones son:

timeout, por defecto 120 segundos, pasado ese tiempo de inactividad se liberan los recursos y deja de "prestar atención" a las conversaciones

max_conversations, por defecto 65535, y es el máximo número de conversaciones simultáneas que puede "*espiar*" al mismo tiempo

allowed_ip_protocols, por defecto todos (all) y permite especificar qué protocolos IP serán monitorizados (TCP, UDP, ICMP, etc...) cada uno de estos protocolos tiene "asociado" un número de protocolo tal y como describen los RFC's correspondientes, lo más usuales son:

TCP = 6, UDP =17, ICMP =1, se puede usar la forma 6 17 1 para los tres a la vez.

alert_odd_protocols, por defecto desactivado, y se generarán alertas cuando se reciban paquetes que no figuren en la directiva anterior (***allowed_ip_protocols***)

Ejemplo

preprocessor conversation: max_conversation 15000, timeout 300, allowed_ip_protocols 1 6 17, alert_odd_protocols

Configurar el preprocesador arpspoof

Detecta las falsificaciones de ***MAC*** en el protocolo ***ARP*** evitando así ataques del tipo ***MiTM*** como los que se enseñaron en el ***número 11 de esta revista***, el artículo de ***moebius*** sobre intrusión en redes locales y ***dsniff.***

Las comunicaciones en una LAN se basan en un ***direccionamiento físico*** (***direcciones MAC***) y cada equipo de la red guarda ***una tabla que relaciona la dirección MAC con la dirección IP*** que le corresponde a los con que se comunica. (***tabla arp***)

Cuando una máquina no conoce la ***MAC*** de otra, lanza una petición ***broadcast*** a toda la red preguntando ***cómo se llama la MAC de la dirección IP con la que quiero "hablar"*** y en condiciones normales, el equipo que tiene la *IP* que se busca responde informando de su *MAC*, se actualizan las tablas *arp* y se procede a la comunicación. En esto se basa el protocolo ARP.

Pero cuando un *usuario malicioso* de la red "*falsifica*" la *MAC* y se hace pasar por el verdadero poseedor de ella, puede interceptar las comunicaciones entre un equipo y otro, si además le añade la posibilidad de reenviar los datos al verdadero destino una vez que hayan pasado por la *máquina maliciosa*, tenemos un bonito escenario llamado "***Hombre en medio***" o "***mono en medio"***

Es decir, los ataques ***MiTM*** se basan en interceptar las comunicaciones entre dos host "*metiéndose por medio*", falsificando la MAC (***envenenamiento ARP***) y reenviando los paquetes una vez "*escuchados*" con lo que el tráfico legítimo entre un *host* y otro se vulnera y pueden capturarse contraseñas de red, correo y otra información sensible de la red.

Para evitar o al menos detectar esos comportamientos, podemos incluir el ***preprocesador arpspoof***, de tal modo que si alguien envenena la red mediante *ARP* saltarán las alertas... y de paso "*le fastidiamos la técnica descrita por moebius en su artículo ya publicado" :P*

El uso de la directiva es muy simple:

preprocessor arpspoof_detect_host: 172.28.0.50 00.05.1c.08.ae.7c
preprocessor arpspoof_detect_host: 172.28.0.25 01.00.c9.af.cd.92

Es decir, **relacionar las direcciones IP con las verdaderas direcciones MAC** de cada equipo que queramos "*proteger*", si alguien en la red intenta utilizar estas direcciones IP o *MAC's* y no se corresponden con la lista mostrada, se disparará una alerta.

Puedes ver la ***MAC*** asignada a un equipo si envías un ***ping*** a su dirección ip y luego consultas la ***tabla arp*** mediante la ***orden arp –a***

ping 172.28.0.1
arp –a

Y si quieres ver tu propia ***mac***, puedes hacerlo con:

ipconfig/all (en windows)
ifconfig (en Linux)

Configurar el pre-procesador fnord

Una forma que snort identifica los ataques "*desconocidos*" es buscando ***shellcodes*** y secuencias *NOP*.

fnord es un *preprocesador* que busca y detecta ***shellcodes polifórmicas***, una ***shellcode*** tiene el objetivo de proporcionar a un atacante una shell del sistema mediante desbordamientos de memoria u *overflows* y es uno de los ataques más devastadores y peligrosos que puedes sufrir, si tiene éxito, reportará una línea de comandos al atacante, o sea, como si estuviese sentado en tú silla, utilizando tú teclado y viendo tú monitor....

Aunque todavía está en desarrollo, es posible usar YA ***fnord***, para ello:

preprocessor fnord

Por este mes hemos terminado... hemos avanzado MUCHO, nos queda los MAS JUGOSO, pero tendrás que esperar al próximo número.... para entonces configuraremos un Sistema de detección de Intrusos con registro de alertas en una Base de Datos y Servicios Web para ver los resultados "*a todo color*", con formatos de salida más legibles y cómodos de interpretar y no tan crudos como los informes que hemos obtenido en este artículo.

Aprenderemos a configurar ***samba*** para comunicarnos con clientes *Windows* y ***Service For Unix*** para comunicarnos con clientes *LiNUX*, enviaremos las alertas por mensajería interna, por mail, hacia una Base de Datos, seremos capaces de utilizar servidores ***syslog,*** y también utilizar otras herramientas para mantener las reglas "*a punto*", vamos todo un elenco de posibilidades para estar informado de TODO lo que pasa por nuestra red.

Y también... cómo no... intentaremos "*saltarnos*" a *snort*.... o al menos haremos toda clase de pruebas de comportamiento (*stress-test*) para ver si nuestras reglas y directivas son apropiadas y tenemos todo, todito, todo bien configurado.